

# I/O-Efficient Dynamic Planar Range Skyline Queries

Casper Kejllberg-Rasmussen  
MADALGO\*  
Department of Computer Science  
Aarhus University, Denmark  
ckr@madalgo.au.dk

Konstantinos Tsakalidis  
Computer Engineering and  
Informatics Department  
University of Patras, Greece  
tsakalid@ceid.upatras.gr

Kostas Tsichlas  
Computer Science Department  
Aristotle University  
of Thessaloniki, Greece  
tsichlas@csd.auth.gr

## Abstract

We present the first fully dynamic worst case I/O-efficient data structures that support planar orthogonal *3-sided range skyline reporting queries* in  $\mathcal{O}(\log_{2B^\epsilon} n + \frac{t}{B^{1-\epsilon}})$  I/Os and updates in  $\mathcal{O}(\log_{2B^\epsilon} n)$  I/Os, using  $\mathcal{O}(\frac{n}{B^{1-\epsilon}})$  blocks of space, for  $n$  input planar points,  $t$  reported points, and parameter  $0 \leq \epsilon \leq 1$ . We obtain the result by extending Sundar’s priority queues with attrition to support the operations DELETEMIN and CATENATEANDATTRITE in  $\mathcal{O}(1)$  worst case I/Os, and in  $\mathcal{O}(1/B)$  amortized I/Os given that a constant number of blocks is already loaded in main memory. Finally, we show that any pointer-based static data structure that supports *dominated maxima reporting queries*, namely the difficult special case of 4-sided skyline queries, in  $\mathcal{O}(\log^{\mathcal{O}(1)} n + t)$  worst case time must occupy  $\Omega(n \frac{\log n}{\log \log n})$  space, by adapting a similar lower bounding argument for planar 4-sided range reporting queries.

## 1 Introduction

We study the problem of maintaining a set of planar points in external memory subject to insertions and deletions of points in order to support planar orthogonal 3-sided range skyline reporting queries efficiently in the worst case. For two points  $p, q \in \mathbb{R}^d$ , we say that  $p$  *dominates*  $q$ , if and only if all the coordinates of  $p$  are greater than those of  $q$ . The *skyline* of a pointset  $P$  consists of the *maximal points* of  $P$ , which are the points in  $P$  that are not dominated by any other point in  $P$ . *Planar 3-sided range skyline reporting queries* that report the maximal points among the points that lie

Skyline computation has been receiving increasing attention in the field of databases since the introduction of the skyline operator for SQL [3]. Skyline points correspond to the “interesting” entries of a relational database as they are optimal simultaneously over all attributes. The considered variant of planar skyline queries adds the capability of reporting the interesting entries among those input entries whose attribute values belong to a given 3-sided range. Databases used in practical applications usually process massive amounts of data in dynamic environments, where the data can be modified by update operations. Therefore we analyze our algorithms in the *I/O model* [1], which is commonly used to capture the complexity of massive data computation. It assumes that the input data resides in the disk (external memory) divided in blocks of  $B$  consecutive words, and that computation occurs for free in the internal memory of size  $M$  words. An *I/O-operation* (*I/O*) reads a block of data from the disk into the internal memory, or writes a block of data

---

\*Center for Massive Data Algorithmics - a Center of the Danish National Research Foundation

to the disk. Time complexity is expressed in number of I/Os, and space complexity in the number of blocks that the input data occupies on the disk.

**Previous Results** Different approaches have been proposed for maintaining the  $d$ -dimensional skyline in external memory under update operations, assuming for example offline updates over data streams [19, 13], only online deletions [20], online average case updates [16], arbitrary online updates [8] and online updates over moving input points [9]. The efficiency of all previous approaches is measured experimentally in terms of disk usage over average case data. However, even for the planar case, no I/O-efficient structure exists that supports both arbitrary insertions and deletions in sublinear worst case I/Os. Regarding internal memory, Brodal and Tsakalidis [4] present two linear space dynamic data structures that support 3-sided range skyline reporting queries in  $\mathcal{O}(\log n + t)$  and  $\mathcal{O}(\frac{\log n}{\log \log n} + t)$  worst case time, and updates in  $\mathcal{O}(\log n)$  and  $\mathcal{O}(\frac{\log n}{\log \log n})$  worst case time in the pointer machine and the RAM model, respectively, where  $n$  is the input size and  $t$  is the output size. They also present an  $\mathcal{O}(n \log n)$  space dynamic pointer-based data structure that supports 4-sided range skyline reporting queries in  $\mathcal{O}(\log^2 n + t)$  worst case time and updates in  $\mathcal{O}(\log^2 n)$  worst case time. Adapting these structures to the I/O model attains  $\mathcal{O}(\log_B^{\mathcal{O}(1)} n + t)$  query I/Os, which is undesired since  $\mathcal{O}(1)$  I/Os are spent per reported point.

Regarding the static variant of the problem, Sheng and Tao [17] obtain an I/O-efficient algorithm that computes the skyline of a static  $d$ -dimensional pointset in  $\mathcal{O}(\frac{n}{B} \log_{\frac{M}{B}}^{d-2} \frac{n}{B})$  worst case I/Os, for  $d \geq 3$ , by adapting the internal memory algorithms of [12, 2] to external memory.  $\mathcal{O}(\frac{n}{B} \log_{\frac{M}{B}} \frac{n}{B})$  I/Os can be achieved for the planar case. There exist two  $\mathcal{O}(n \log n)$  and  $\mathcal{O}(n \frac{\log n}{\log \log n})$  space static data structures that support planar 4-sided range skyline reporting queries in  $\mathcal{O}(\log n + t)$  and  $\mathcal{O}(\frac{\log n}{\log \log n} + t)$  worst case time, for the pointer machine and the RAM, respectively [10, 7].

**Our Results** In Section 3 we present the basic building block of the structures for dynamic planar range skyline reporting queries that we present in Section 4. That is pointer-based *I/O-efficient catenable priority queues with attrition (I/O-CPQAs)* that support the operations DELETEMIN and CATENATEANDATTRITE in  $\mathcal{O}(1/B)$  amortized I/Os and in  $\mathcal{O}(1)$  worst case I/Os, using  $\mathcal{O}(\frac{n-m}{B})$  disk blocks, after  $n$  calls to CATENATEANDATTRITE and  $m$  calls to DELETEMIN. The result is obtained by modifying appropriately a proposed implementation for priority queues with attrition of Sundar [18].

In Section 4 we present our main result, namely I/O-efficient dynamic data structures that support 3-sided range skyline reporting queries in  $\mathcal{O}(\log_{2B^\epsilon} n + \frac{t}{B^{1-\epsilon}})$  worst case I/Os and updates in  $\mathcal{O}(\log_{2B^\epsilon} n)$  worst case I/Os, using  $\mathcal{O}(\frac{n}{B^{1-\epsilon}})$  blocks, for a parameter  $0 \leq \epsilon \leq 1$ . These are the first fully dynamic skyline data structures for external memory that support all operations in polylogarithmic worst case time. The results are obtained by following the approach of Overmars and van Leeuwen [14] for planar skyline maintenance and utilizing confluent persistent I/O-CPQAs (implemented with functional catenable deques [11]). Applying the same methodology to internal memory pointer-based CPQAs yields alternative implementations for dynamic 3-sided reporting in the pointer machine in the same bounds as in [4].

Finally, in Section 5 we prove that any pointer-based static data structure that supports reporting the maximal points among the points that are dominated by a given query point in  $\mathcal{O}(\log^{\mathcal{O}(1)} n)$  worst case time must occupy  $\Omega(n \frac{\log n}{\log \log n})$  space, by adapting the similar lower bounding argument of Chazelle [5] for planar 4-sided range reporting queries to the considered dominated skyline reporting queries. These queries are termed as *dominating minima reporting queries*. The symmetric case of *dominated maxima reporting queries* is equivalent and comprises a special case of rectangular visibility queries [15] and 4-sided range skyline reporting queries [4, 10]. The result shows that the space usage of the pointer-based structures in [15, 4, 10] is optimal within a  $\mathcal{O}(\log \log n)$  factor, for the attained query time.

## 2 Preliminaries

**Priority Queues with Attrition** Sundar [18] introduces pointer-based *priority queues with attrition (PQAs)* that support the following operations in  $\mathcal{O}(1)$  worst case time on a set of elements drawn from a total

order: DELETEMIN deletes and returns the minimum element from the PQA, and INSERTANDATTRITE( $e$ ) inserts element  $e$  into the PQA and removes all elements larger than  $e$  from the PQA. PQAs use space linear to the number of inserted elements minus the number of elements removed by DELETEMIN.

**Functional Catenable Deques** A dynamic data structure is *persistent* when it maintains its previous versions as update operations are performed on it. It is *fully persistent* when it permits accessing and updating the previous versions. In turn, it is called *confluently persistent* when it is fully persistent, and moreover it allows for two versions to be combined into a new version, by use of an update operation that merges the two versions. In this case, the versions form a directed acyclic version graph. A catenable deque is a list that stores a set of elements from a total order, and supports the operations PUSH and INJECT that insert an element to the head and tail of the list respectively, POP and EJECT that remove the element from the head and tail of the list respectively, and CATENATE that concatenates two lists into one. Kaplan and Tarjan [11] present *purely functional catenable deques* that are confluently persistent and support the above operations in  $\mathcal{O}(1)$  worst case time.

**Searching Lower Bound in the Pointer Machine** In the pointer machine model a data structure that stores a data set  $S$  and supports range reporting queries for a query set  $\mathcal{Q}$ , can be modelled as a directed graph  $G$  of bounded out-degree. In particular, every node in  $G$  may be assigned an element of  $S$  or may contain some other useful information. For a query range  $Q_i \in \mathcal{Q}$ , the algorithm navigates over the edges of  $G$  in order to locate all nodes that contain the answer to the query. The algorithm may also traverse other nodes. The time complexity of reporting the output of  $Q_i$  is at least equal to the number of nodes accessed in graph  $G$  for  $Q_i$ . To prove a lower bound we need to construct hard instances with particular properties, as discussed by Chazelle and Liu [5, 6]. In particular, they define the graph  $G$  to be  $(\alpha, \omega)$ -*effective*, if a query is supported in  $\alpha(t + \omega)$  time, where  $t$  is the output size,  $\alpha$  is a multiplicative factor for the output size ( $\alpha = \mathcal{O}(1)$  for our purposes) and  $\omega$  is the additive factor. They also define a query set  $\mathcal{Q}$  to be  $(m, \omega)$ -*favorable* for a data set  $S$ , if  $|S \cap Q_i| \geq \omega, \forall Q_i \in \mathcal{Q}$  and  $|S \cap Q_{i_1} \cap \dots \cap Q_{i_m}| = \mathcal{O}(1), \forall i_1 < i_2 < \dots < i_m$ . Intuitively, the first part of this property requires that the size of the output is large enough (at least  $\omega$ ) so that it dominates the additive factor of  $\omega$  in the time complexity. The second part requires that the query outputs have minimum overlap, in order to force  $G$  to be large without many nodes containing the output of many queries. The following lemma exploits these properties to provide a lower bound on the minimum size of  $G$ .

**Lemma 2.1.** [6, Lemma 2.3] *For an  $(m, \omega)$ -favorable graph  $G$  for the data set  $S$ , and for an  $(\alpha, \omega)$ -effective set of queries  $\mathcal{Q}$ ,  $G$  contains  $\Omega(|\mathcal{Q}|\omega/m)$  nodes, for constant  $\alpha$  and for any large enough  $\omega$ .*

### 3 I/O-Efficient Catenable Priority Queues with Attrition

In this Section, we present I/O-efficient *catenable priority queues with attrition (I/O-CPQAs)* that store a set of elements from a total order in external memory, and support the following operations:

FINDMIN( $Q$ ) returns the minimum element in I/O-CPQA  $Q$ .

DELETEMIN( $Q$ ) removes the minimum element  $e$  from I/O-CPQA  $Q$  and returns element  $e$  and the new I/O-CPQA  $Q' = Q \setminus \{e\}$ .

CATENATEANDATTRITE( $Q_1, Q_2$ ) concatenates I/O-CPQA  $Q_2$  to the end of I/O-CPQA  $Q_1$ , removes all elements in  $Q_1$  that are larger than the minimum element in  $Q_2$ , and returns a new I/O-CPQA  $Q'_1 = \{e \in Q_1 | e < \min(Q_2)\} \cup Q_2$ . We say that the removed elements have been *attrited*.

INSERTANDATTRITE( $Q, e$ ) inserts element  $e$  at the end of  $Q$  and attrites all elements in  $Q$  that are larger than the value of  $e$ .

All operations take  $\mathcal{O}(1)$  worst case I/Os and  $\mathcal{O}(1/b)$  amortized I/Os, given that a constant number of blocks is already loaded into main memory, for a parameter  $1 \leq b \leq B$ . To achieve the result, we modify an implementation for the PQAs of Sundar [18].

An I/O-CPQA  $Q$  consists of  $k_Q + 2$  deques of records, called the clean deque  $C(Q)$ , the buffer deque  $B(Q)$  and the dirty deques  $D_1(Q), \dots, D_{k_Q}(Q)$ , where  $k_Q \geq 0$ . A record  $r = (l, p)$  consists of a buffer  $l$  of  $[b, 4b]$  elements of strictly increasing value and a pointer  $p$  to an I/O-CPQA. The ordering of  $r$  is; first all elements of  $l$  and then all elements of the I/O-CPQA pointed to by  $p$ . We define the queue order of  $Q$  to be  $C(Q)$ ,  $B(Q)$  and  $D_1(Q), \dots, D_{k_Q}(Q)$ . A record is *simple* when its pointer  $p$  is *null*. The clean deque and the buffer deque only contains simple records. See Figure 1 for an overview of the structure.

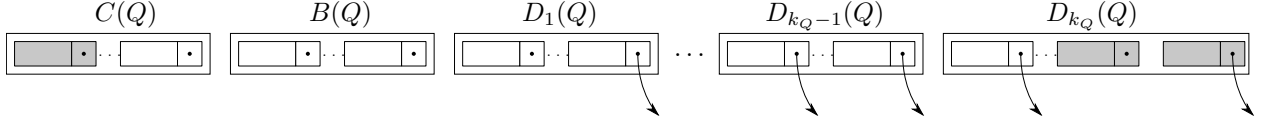


Figure 1: A I/O CPQA  $Q$  consists of  $k_Q + 2$  deques of records;  $C(Q), B(Q), D_1(Q), \dots, D_{k_Q}(Q)$ . The records in  $C(Q)$  and  $B(Q)$  are simple, the records of  $D_1(Q), \dots, D_{k_Q}(Q)$  may contain pointers to other I/O CPQA's. Gray records are always loaded in memory.

Given a record  $r = (l, p)$  the minimum and maximum elements in the buffers of  $r$ , are denoted by  $\min(r) = \min(l)$  and  $\max(r) = \max(l)$ , respectively. They appear respectively first and last in the queue order of  $l$ , since the buffer of  $r$  is sorted by value. Henceforth, we do not distinguish between an element and its value. Given a deque  $q$  the first and the last record is denoted by  $\text{first}(q)$  and  $\text{last}(q)$ , respectively. Also  $\text{rest}(q)$  denotes all records of the deque  $q$  excluding the record  $\text{first}(q)$ . Similarly,  $\text{front}(q)$  denotes all records for the deque  $q$  excluding the record  $\text{last}(q)$ . The size  $|r|$  of a record  $r$  is defined to be the number of elements in its buffer. The size  $|q|$  of a deque  $q$  is defined to be the number of records it contains. The size  $|Q|$  of the I/O-CPQA  $Q$  is defined to be the number of elements that  $Q$  contains. For an I/O-CPQA  $Q$  we denote by  $\text{first}(Q)$  and  $\text{last}(Q)$ , the first and last of the records in  $C(Q), B(Q), D_1(Q), \dots, D_{k_Q}(Q)$  that exists, respectively. By  $\text{middle}(Q)$  we denote all records in  $Q$  and the records in the I/O-CPQAs pointed by  $Q$ , except for records  $\text{first}(Q)$  and  $\text{last}(Q)$  and the I/O-CPQAs they point to. We call an I/O-CPQA  $Q$  *large* if  $|Q| \geq b$  and *small* otherwise. The minimum value of all elements stored in the I/O-CPQA  $Q$  is denoted by  $\min(Q)$ . For an I/O-CPQA  $Q$  we maintain the following invariants:

- I.1) For every record  $r = (l, p)$  where pointer  $p$  points to I/O-CPQA  $Q'$ ,  $\max(l) < \min(Q')$  holds.
- I.2) In all deques of  $Q$ , where record  $r_1 = (l_1, p_1)$  precedes record  $r_2 = (l_2, p_2)$ ,  $\max(l_1) < \min(l_2)$  holds.
- I.3) For the deques  $C(Q), B(Q)$  and  $D_1(Q)$ ,  $\max(\text{last}(C(Q))) < \min(\text{first}(B(Q))) < \min(\text{first}(D_1(Q)))$  holds.
- I.4) Element  $\min(\text{first}(D_1(Q)))$  has the minimum value among all the elements in the dirty deques  $D_1(Q), \dots, D_k(Q)$ .
- I.5) All records in the deques  $C(Q)$  and  $B(Q)$  are simple.
- I.6)  $|C(Q)| \geq \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q - 1$ .
- I.7)  $|\text{first}(C(Q))| < b$  holds, if and only if  $|Q| < b$  holds.
- I.8)  $|\text{last}(D_{k_Q}(Q))| < b$  holds, if and only if record  $\text{last}(D_{k_Q}(Q))$  is simple. In this case  $|r| \in [b, 5b]$  holds.

From Invariants I.2, I.3 and I.4, we have that the minimum element  $\min(Q)$  stored in the I/O-CPQA  $Q$  is element  $\min(\text{first}(C(Q)))$ . We say that an operation *improves* or *aggravates* by a parameter  $c$  the inequality of invariant I.6 for I/O-CPQA  $Q$ , when the operation increases or decreases  $\Delta(Q) = |C(Q)| - \sum_{i=1}^{k_Q} |D_i(Q)| - k_Q + 1$  by  $c$ , respectively. To argue about the  $\mathcal{O}(1/b)$  amortized I/O bounds we define the following potential

functions for large and small I/O-CPQAs. In particular, for large I/O-CPQAs  $Q$ , the potential  $\Phi(Q)$  is defined as

$$\Phi(Q) = \Phi_F(|\text{first}(Q)|) + |\text{middle}(Q)| + \Phi_L(|\text{last}(Q)|),$$

where

$$\Phi_F(x) = \begin{cases} 3 - \frac{x}{b}, & b \leq x < 2b \\ 1, & 2b \leq x < 3b \\ \frac{2x}{b} - 5, & 3b \leq x \leq 4b \end{cases} \quad \text{and} \quad \Phi_L(x) = \begin{cases} 0, & 0 \leq x < 4b \\ \frac{3x}{b} - 12, & 4b \leq x \leq 5b \end{cases}$$

For small I/O-CPQAs  $Q$ , the potential  $\Phi(Q)$  is defined as

$$\Phi(Q) = \frac{3|Q|}{b}$$

The total potential  $\Phi_T$  is defined as

$$\Phi_T = \sum_Q \Phi(Q) + \sum_{Q|b \leq |Q|} 1,$$

where the first sum is over all I/O-CPQAs  $Q$  and the second sum is only over all large I/O-CPQAs  $Q$ .

### 3.1 Operations

In the following, we describe the algorithms that implement the operations supported by the I/O-CPQA  $Q$ . The operations call the auxiliary operation  $\text{BIAS}(Q)$ , which will be described last, that improves the inequality of invariant I.6 for  $Q$  by at least 1. All operations take  $\mathcal{O}(1)$  worst case I/Os. We also show that every operation takes  $\mathcal{O}(1/b)$  amortized I/Os, where  $1 \leq b \leq B$ .

**FindMin**( $Q$ ) returns the value  $\min(\text{first}(C(Q)))$ .

**DeleteMin**( $Q$ ) removes element  $e = \min(\text{first}(C(Q)))$  from record  $(l, p) = \text{first}(C(Q))$ . After the removal, if  $|l| < b$  and  $|Q| \geq b$  hold, we do the following. If  $b \leq |\text{first}(\text{rest}(C(Q)))| \leq 2b$ , then we merge  $\text{first}(C(Q))$  with  $\text{first}(\text{rest}(C(Q)))$  into one record which is the new first record. Else if  $2b < |\text{first}(\text{rest}(C(Q)))| \leq 3b$  then we take  $b$  elements out of  $\text{first}(\text{rest}(C(Q)))$  and put them into  $\text{first}(C(Q))$ . Else we have that  $3b < |\text{first}(\text{rest}(C(Q)))|$ , and as a result we take  $2b$  elements out of  $\text{first}(\text{rest}(C(Q)))$  and put them into  $\text{first}(C(Q))$ . If the inequality for  $Q$  is aggravated by 1 we call  $\text{BIAS}(Q)$  once. Finally, element  $e$  is returned.

*Amortization:* Only if the size of  $\text{first}(C(Q))$  becomes  $|\text{first}(C(Q))| = b - 1$  do we incur any I/Os. In this case  $r = \text{first}(Q)$  has a potential of  $\Phi_F(|r|) = 2$ , and since we increase the number of elements in  $r$  by  $b$  to  $2b$  elements, the potential of  $r$  will then only be  $\Phi_F(|r|) = 1$ . Thus, the total potential decreases by 1, which also pays for any I/Os including those incurred if  $\text{BIAS}(Q)$  is invoked.

**CatenateAndAttrite**( $Q_1, Q_2$ ) concatenates  $Q_2$  to the end of  $Q_1$  and removes the elements from  $Q_1$  with value larger than  $\min(Q_2)$ . To do so, it creates a new I/O-CPQA  $Q'_1$  by modifying  $Q_1$  and  $Q_2$ , and by calling  $\text{BIAS}(Q'_1)$  and  $\text{BIAS}(Q_2)$ .

If  $|Q_1| < b$ , then  $Q_1$  is only one record  $(l_1, \cdot)$ , and so we prepend it into the first record  $(l_2, \cdot) = \text{first}(Q_2)$  of  $Q_2$ . Let  $l'_1$  be the non-attrited elements of  $l_1$ . We perform the prepend as follows. If  $|l'_1| + |l_2| \leq 4b$ , then we prepend  $l'_1$  into  $l_2$ . Else, we take  $2b - |l'_1|$  elements out of  $l_2$ , and make them along with  $l'_1$  the new first record of  $Q_2$ .

*Amortization:* If we simply prepend  $l'_1$  into  $l_2$ , then the potential  $\Phi_S(|l_1|)$  pays for the increase in potential of  $\Phi_F(|\text{first}(C(Q_2))|)$ . Else, we take  $2b - |l'_1|$  elements out of  $l_2$ , and these elements along with  $l'_1$  become the new first record of  $Q_2$  of size  $2b$ . Thus,  $\Phi_F(2b) = 1$  and the potential drops by 1, which is enough to pay for the I/Os used to flush the old first record of  $C(Q_2)$  to disk.

If  $|Q_2| < b$ , then  $Q_2$  only consists of one record. We have two cases, depending on how much of  $Q_1$  is attrited by  $Q_2$ . Let  $r_1$  be the second last record for  $Q_1$  and let  $r_2 = \text{last}(Q_1)$  be the last record. If  $e$  attrites all of

$r_1$ , then we just pick the appropriate case among (1–4) below. Else if  $e$  attrites partially  $r_1$ , but not all of it, then we delete  $r_2$  and we merge  $r_1$  and  $Q_2$  into the new last record of  $Q_1$ , which cannot be larger than  $5b$ . Otherwise if  $e$  attrites partially  $r_2$ , but not all of it, then we simply append the single record of  $Q_2$  into  $r_2$ , which will be the new last record of  $Q_1$  and it cannot be larger than  $5b$ .

*Amortization:* If  $e$  attrites all of  $r_1$ , then we release at least 1 in potential, so all costs in any of the cases (1–4) are paid for. If  $e$  attrites partially  $r_1$ , then the new record cannot contain more than  $5b$  elements, and thus any increase in potential is paid for by the potential of  $Q_2$ . Thus, the I/O cost is covered by the decrease of 1 in potential, caused by  $r_1$ . If  $e$  attrites partially  $r_2$ , any increase in potential is paid for by the potential of  $Q_2$ .

We have now dealt with the case where  $Q_1$  is a small queue, so in the following we assume that  $Q_1$  is large. Let  $e = \min(Q_2)$ .

- 1) If  $e \leq \min(\text{first}(C(Q_1)))$ , we discard I/O-CPQA  $Q_1$  and set  $Q'_1 = Q_2$ .
- 2) Else if  $e \leq \max(\text{last}(C(Q_1)))$ , we remove the simple record  $(l, \cdot) = \text{first}(C(Q_2))$  from  $C(Q_2)$ , we set  $C(Q'_1) = \emptyset$ ,  $B(Q'_1) = C(Q_1)$  and  $D_1(Q'_1) = (l, p)$ , where  $p$  points to  $Q_2$ , if it exists. This aggravates the inequality for  $Q_2$  by at most 1, and gives  $\Delta(Q'_1) = -1$ . Thus, we call  $\text{BIAS}(Q_2)$  once and  $\text{BIAS}(Q'_1)$  once.
- 3) Else if  $e \leq \min(\text{first}(B(Q_1)))$  or  $e \leq \min(\text{first}(D_1(Q_1)))$  holds, we remove the simple record  $(l, \cdot) = \text{first}(C(Q_2))$  from  $C(Q_2)$ , set  $D_1(Q'_1) = (l, p)$ , and make  $p$  point to  $Q_2$ , if it exists. If  $e \leq \min(\text{first}(B(Q_1)))$ , we set  $B(Q'_1) = \emptyset$ . This aggravates the inequality for  $Q_2$  by at most 1, and aggravates the inequality for  $Q_1$  by at most 1. Thus, we call  $\text{BIAS}(Q_2)$  once and  $\text{BIAS}(Q'_1)$  once.
- 4) Else, let  $(l_1, \cdot) = \text{last}(D_{k_{Q_1}})$ . We remove  $(l_2, \cdot) = \text{first}(C(Q_2))$  from  $C(Q_2)$ . If  $|l_1| < b$ , then remove the record  $(l_1, \cdot)$  from  $D_{k_{Q_1}}$ . Let  $l'_1$  be the non-attrited elements under attrition by  $e = \min(l_2)$ . If  $|l'_1| + |l_2| \leq 4b$ , then we prepend  $l'_1$  into  $l_2$  of record  $r_2 = (l_2, p_2)$ , where  $p_2$  points to  $Q_2$ . Otherwise, we make a new simple record  $r_1$  with  $l'_1$  and  $2b$  elements taken out of  $r_2 = (l_2, p_2)$ . Finally, we put the resulting one or two records  $r_1$  and  $r_2$  into a new deque  $D_{k_{Q_1}+1}(Q_1)$ . This aggravates the inequality for  $Q_2$  by at most 1, and the inequality for  $Q_1$  by at most 2. Thus, we call  $\text{BIAS}(Q_2)$  once and  $\text{BIAS}(Q'_1)$  twice.

*Amortization:* In all the cases (1–4) both  $Q_1$  and  $Q_2$  are large, hence when we concatenate them we decrease the potential by at least 1, as the number of large I/O-CPQA's decrease by one which is enough to pay for any  $\text{BIAS}$  operations.

**InsertAndAttrite** $(Q, e)$  inserts an element  $e$  into I/O-CPQA  $Q$  and attrites the elements in  $Q$  with value larger than  $e$ . This is a special case of operation  $\text{CATENATEANDATTRITE}(Q_1, Q_2)$ , where  $Q_1 = Q$  and  $Q_2$  is an I/O-CPQA that only contains one record with the single element  $e$ .

*Amortization:* Since creating a new I/O-CPQA with only one element and calling  $\text{CATENATEANDATTRITE}$  only costs  $\mathcal{O}(1/b)$  I/Os amortized, the operation  $\text{INSERTANDATTRITE}$  also costs  $\mathcal{O}(1/b)$  I/Os amortized.

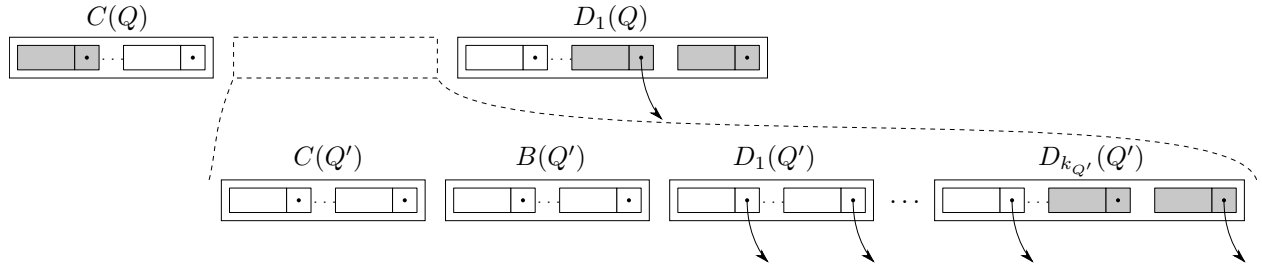


Figure 2: In the case of  $\text{BIAS}(Q)$ , where  $B(Q) = \emptyset$  and  $k_Q = 1$ , we need to follow the pointer  $p$  of  $(l, p) = \text{first}(D_1(Q))$  that may point to an I/O-CPQA  $Q'$ . If so, we merge it into  $Q$ , taking into account attrition of  $Q'$  by  $e = \min(\text{first}(D_1(Q)))$ .

**Bias**( $Q$ ) improves the inequality in I.6 for  $Q$  by at least 1.

*Amortization:* Since all I/Os incurred by **BIAS**( $Q$ ) are already paid for by the operation that called **BIAS**( $Q$ ), we only need to argue that the potential of  $Q$  does not increase due to the changes that **BIAS**( $Q$ ) makes to  $Q$ .

1)  $|B(Q)| > 0$ : We remove the first record  $\text{first}(B(Q)) = (l_1, \cdot)$  from  $B(Q)$  and let  $(l_2, p_2) = \text{first}(D_1(Q))$ . Let  $l'_1$  be the non-attributed elements of  $l_1$  under attrition from  $e = \min(l_2)$ .

- 1)  $0 \leq |l'_1| < b$ : If  $|l_2| \leq 2b$ , then we just prepend  $l'_1$  onto  $l_2$ . Else, we take  $b$  elements out of  $l_2$  and append them to  $l'_1$ .
- 2)  $b \leq |l'_1| < 2b$ : If  $|l_2| \leq 2b$ , and if furthermore  $|l'_1| + |l_2| \leq 3b$  holds, then we merge  $l'_1$  and  $l_2$ . Else  $|l'_1| + |l_2| > 3b$  holds, so we take  $2b$  elements out of  $l'_1$  and  $l_2$  and put them into  $l'_1$ , leaving the rest in  $l_2$ .  
Else  $|l_2| > 2b$  holds, so we take  $b$  elements out of  $l_2$  and put them into  $l'_1$ .

If we did not prepend  $l'_1$  onto  $l_2$ , we insert  $l'_1$  along with any elements taken out of  $l_2$  at the end of  $C(Q)$  instead. If  $|l'_1| < |l_1|$ , we set  $B(Q) = \emptyset$ . Else, we did prepend  $l'_1$  onto  $l_2$ , and then we just recursively call **BIAS**. Since  $|B(Q)| = 0$  we will not end up in this case again. As a result, in all cases the inequality of  $Q$  is improved by 1.

*Amortization:* If  $l_1 = \text{first}(Q)$ , then after calling **BIAS** we ensure that  $2b \leq |\text{first}(Q)| \leq 3b$ , and so the that potential of  $Q$  does not increase.

2)  $|B(Q)| = 0$ : When  $|B(Q)| = 0$  holds, we have two cases depending on the number of dirty queues, namely cases  $k_Q > 1$  and  $k_Q = 1$ .

1)  $k_Q > 1$ : Let  $e = \min(\text{first}(D_{k_Q}(Q)))$ . If  $e \leq \min(\text{last}(D_{k_Q-1}(Q)))$  holds, we remove the record  $\text{last}(D_{k_Q-1}(Q))$  from  $D_{k_Q-1}(Q)$ . This improves the inequality of  $Q$  by 1.

Else, if  $\min(\text{last}(D_{k_Q-1}(Q))) < e \leq \max(\text{last}(D_{k_Q-1}(Q)))$  holds, we remove record  $r_1 = (l_1, p_1) = \text{last}(D_{k_Q-1}(Q))$  from  $D_{k_Q-1}(Q)$  and let  $r_2 = (l_2, p_2) = \text{first}(D_{k_Q}(Q))$ . We delete any elements in  $l_1$  that are attributed by  $e$ , and let  $l'_1$  denote the non-attributed elements.

- 1)  $0 \leq |l'_1| < b$ : If  $|l_2| \leq 2b$ , then we just prepend  $l'_1$  onto  $l_2$ . Otherwise, we take  $b$  elements out of  $l_2$  and append them to  $l'_1$ .
- 2) If  $b \leq |l'_1| < 2b$ : If  $|l_2| \leq 2b$  and  $|l'_1| + |l_2| \leq 3b$ , then we merge  $l'_1$  and  $l_2$ . Else,  $|l'_1| + |l_2| > 3b$  holds, so we take  $2b$  elements out of  $l'_1$  and  $l_2$  and put them into  $l'_1$ , leaving the rest in  $l_2$ .  
Else  $|l_2| > 2b$ , so we take  $b$  elements out of  $l_2$  and put them into  $l'_1$ .

If  $r_1$  still exists, we insert it in the front of  $D_{k_Q}(Q)$ . Finally, we concatenate  $D_{k_Q-1}(Q)$  and  $D_{k_Q}(Q)$  into one deque. This improves the inequality of  $Q$  by at least 1.

Else  $\max(\text{last}(D_{k_Q-1}(Q))) < e$  holds, and we just concatenate the deques  $D_{k_Q-1}(Q)$  and  $D_{k_Q}(Q)$ , which improves the inequality for  $Q$  by 1.

*Amortization:* If not all of  $l_1$  is attributed then we ensure that its record  $r_1$  has size between  $2b$  and  $3b$ . Thus, if  $r_1 = \text{first}(Q)$  holds, we will not have increased the potential of  $Q$ . In the cases where all or none of  $l_1$  is attributed, the potential of  $Q$  can only be decreased by at least 0.

2)  $k_Q = 1$ : In this case  $Q$  contains only deques  $C(Q)$  and  $D_1(Q)$ . We remove the record  $r = (l, p) = \text{first}(D_1(Q))$  and insert  $l$  into a new record at the end of  $C(Q)$ . This improves the inequality of  $Q$  by at least 1. If  $r$  is not simple, let  $r$ 's pointer  $p$  point to I/O-CPQA  $Q'$ . We restore I.5 for  $Q$  by merging I/O-CPQAs  $Q$  and  $Q'$  into one I/O-CPQA. See Figure 2 for this case of operation **BIAS**. In particular, let  $e = \min(\text{first}(D_1(Q)))$ , we now proceed as follows:

If  $e \leq \min(Q')$ , we discard  $Q'$ . The inequality for  $Q$  remains unaffected.

Else, if  $\min(\text{first}(C(Q'))) < e \leq \max(\text{last}(C(Q')))$ , we set  $B(Q) = C(Q')$  and discard the rest of  $Q'$ . The inequality for  $Q$  remains unaffected.

Else if  $\max(\text{last}(C(Q')) < e \leq \min(\text{first}(D_1(Q')))$ , we concatenate the deque  $C(Q')$  at the end of  $C(Q)$ . If moreover  $\min(\text{first}(B(Q'))) < e$  holds, we set  $B(Q) = B(Q')$ . Finally, we discard the rest of  $Q'$ . This improves the inequality for  $Q$  by  $|C(Q')|$ .

Else  $\min(\text{first}(D_1(Q'))) < e$  holds. We concatenate the deque  $C(Q')$  at the end of  $C(Q)$ , we set  $B(Q) = B(Q')$ , we set  $D_1(Q'), \dots, D_{k_{Q'}}(Q')$  as the first  $k_{Q'}$  dirty queues of  $Q$  and we set  $D_1(Q)$  as the last dirty queue of  $Q$ . This improves the inequality for  $Q$  by  $\Delta(Q') \geq 0$ , since  $Q'$  satisfied I.6 before the operation.

If  $r = \text{first}(Q)$  and  $|l| \leq 2b$ , then we remove  $r$  and run BIAS recursively. Let  $r' = (l', p') = \text{first}(Q)$ . If  $|l| + |l'| > 3b$ , then we take the  $2b$  first elements out and make them the new first record of  $C(Q)$ . Else we merge  $l$  into  $l'$ , so that  $r$  is removed and  $r'$  is now  $\text{first}(Q)$ .

*Amortization:* Since  $\text{first}(Q)$  is either untouched or left with  $2b$  to  $3b$  elements, in which case its potential is 1, and since all other changes decrease the potential by at least 0, we have that BIAS does not increase the potential of  $Q$ .

**Theorem 3.1.** *A set of  $\ell$  I/O-CPQA's can be maintained supporting the operations FINDMIN, DELETEMIN, CATENATEANDATTRITE and INSERTANDATTRITE in  $\mathcal{O}(1/b)$  I/Os amortized and  $\mathcal{O}(1)$  worst case I/Os per operation. The space usage is  $\mathcal{O}(\frac{n-m}{b})$  blocks after calling CATENATEANDATTRITE and INSERTANDATTRITE  $n$  times and DELETEMIN  $m$  times, respectively. We require that  $M \geq \ell b$  for  $1 \leq b \leq B$ , where  $M$  is the main memory size and  $B$  is the block size.*

*Proof.* The correctness follows by closely noticing that we maintain invariants I.1–I.8, and from those we have that DELETEMIN( $Q$ ) and FINDMIN( $Q$ ) always returns the minimum element of  $Q$ .

The worst case I/O bound of  $\mathcal{O}(1)$  is trivial as every operation only touches  $\mathcal{O}(1)$  records. Although BIAS is recursive, we notice that in the case where  $|B(Q)| > 0$ , BIAS only calls itself after making  $|B(Q)| = 0$ , so it will not end up in this case again. Similarly, if  $|B(Q)| = 0$  and  $k_Q > 1$  there might also be a recursive call to BIAS. However, before the call at least  $b$  elements have been taken out of  $Q$ , and thus the following recursive call to BIAS will ensure at least  $b$  more are taken out. This is enough to stop the recursion, which will have depth at most 3.

The  $\mathcal{O}(1/b)$  amortized I/O bounds, follows from the potential analysis made throughout the description of each operation.  $\square$

### 3.2 Concatenating a Sequence of I/O-CPQAs

We describe how to CATENATEANDATTRITE I/O-CPQAs  $Q_1, Q_2, \dots, Q_\ell$  into a single I/O-CPQA in  $\mathcal{O}(1)$  worst case I/Os, given that DELETEMIN is not called in the sequence of operations. We moreover impose two more assumptions. In particular, we say that I/O-CPQA  $Q$  is in state  $x \in \mathbb{Z}$ , if  $|C(Q)| = \sum_{i=1}^{k_Q} |D_i(Q)| + k_Q - 1 + x$  holds. Positive  $x$  implies that BIAS( $Q$ ) will be called after the inequality for  $Q$  is aggravated by  $x + 1$ . Negative  $x$  implies that BIAS( $Q$ ) need to be called  $x$  operations times in order to restore inequality for  $Q$ . So, we moreover assume that I/O-CPQAs  $Q_i, i \in [1, \ell]$  are at state at least  $+2$ , unless  $Q_i$  contains only one record in which case it may be in state  $+1$ . We call a record  $r = (l, p)$  in an I/O-CPQA  $Q_i$  *critical*, if  $r$  is accessed at some time during the sequence of operations. In particular, the critical records for  $Q_i$  are  $\text{first}(C(Q_i)), \text{first}(\text{rest}(C(Q_i))), \text{last}(C(Q_i)), \text{first}(B(Q_i)), \text{first}(D_1(Q_i)), \text{last}(D_{k_{Q_i}}(Q_i))$ , and  $\text{last}(\text{front}(D_{k_{Q_i}}(Q_i)))$  if it exists. Otherwise, record  $\text{last}(D_{k_{Q_i}-1}(Q_i))$  is critical. So, we moreover assume that the critical records for I/O-CPQAs  $Q_i, i \in [1, \ell]$  are loaded into memory.

The algorithm considers I/O-CPQAs  $Q_i$  in decreasing index  $i$  (from right to left). It sets  $Q^i = Q_\ell$  and constructs the temporary I/O-CPQA  $Q^{i-1}$  by calling CATENATEANDATTRITE( $Q_{i-1}, Q^i$ ). This yields the final I/O-CPQA  $Q^1$ .

**Lemma 3.1.** *I/O-CPQAs  $Q_i, i \in [1, \ell]$  can be CATENATEANDATTRITED into a single I/O-CPQA without any access to external memory, provided that:*

1.  $Q_i$  is in state at least  $+2$ , unless it contains only one record, in which case its state is at least  $+1$ ,



2. all critical records of all  $Q_i$  reside in main memory.

*Proof.* To avoid any I/Os during the sequence of CATENATEANDATTRITES, we ensure that BIAS is not called, and that the critical records are sufficient, and thus no more records need to be loaded into memory.

To avoid calling BIAS we prove by induction the invariant that the temporary I/O-CPQAs  $Q^i, i \in [1, \ell]$  constructed during the sequence are in state at least  $+1$ . Let the invariant hold of  $Q^{i+1}$  and let  $Q^i$  be constructed by CATENATEANDATTRITE( $Q_i, Q^{i+1}$ ). If  $Q_i$  contains at most two records, which both reside in dequeue  $C(Q_i)$ , we only need to access record  $\text{first}(C(Q^{i+1}))$  and the at most two records of  $Q_i$ . The invariant holds for  $Q^i$ , since it holds inductively for  $Q^{i+1}$  and the new records were added at  $C(Q^{i+1})$ . As a result, the inequality of I.6 for  $Q^{i+1}$  can only be improved. If  $Q^{i+1}$  consists of only one record, then either one of the following cases apply or we follow the steps described in operation CATENATEANDATTRITE. In the second case, there is no aggravation for the inequality of 6 and only critical records are used.

In the following, we can safely assume that  $Q_i$  has at least three records and its state is at least  $+2$ . We parse the cases of the CATENATEANDATTRITE algorithm assuming that  $e = \min(Q^{i+1})$ .

- Case 1 The invariant holds trivially since  $Q_i$  is discarded and no change happens to  $Q^i = Q^{i+1}$ . BIAS is not called.
- Cases 2,3 The algorithm checks whether the first two records of  $C(Q_i)$  are attrited by  $e$ . If this is the case, we continue as denoted at the start of this proof. Otherwise, case 2 of CATENATEANDATTRITE is applied as is.  $Q^{i+1}$  is in state 0 after the concatenation and  $Q^i$  is in state  $+1$ . Thus the invariant holds, and BIAS is not. Note that all changes take place at the critical records of  $Q_i$  and  $Q^{i+1}$ .
- Case 4 The algorithm works exactly as in case 4 of CATENATEANDATTRITE, with the following exception. At the end,  $Q^i$  will be in state 0, since we added the deque  $D_{k_{Q^{i+1}}+1}$  with a new record and the inequality of I.6 is aggravated by 2. To restore the invariant we apply case 2(1) of BIAS. This step requires access to records  $\text{last}(D_{k_{Q^i}-1})$  and  $\text{first}(D_{k_{Q^i}})$ . These records are both critical, since the former corresponds to  $\text{last}(D_{k_{Q^{i+1}}})$  and the latter to  $\text{first}C(Q^{i+1})$ . In addition, BIAS( $Q^{i+1}$ ) need not be called, since by the invariant,  $Q^{i+1}$  was in state  $+1$  before the removal of  $\text{first}C(Q^{i+1})$ . In this way, we improve the inequality for  $Q^i$  by 1 and invariant holds.

□

## 4 Dynamic Planar Range Skyline Reporting

In this Section we present dynamic I/O-efficient data structures that support 3-sided planar orthogonal range skyline reporting queries.

**3-Sided Skyline Reporting** We describe how to utilize I/O-CPQAs in order to obtain dynamic data structures that support 3-sided range skyline reporting queries and arbitrary insertions and deletions of points, by modifying the approach of [14] for the pointer machine model. In particular, let  $P$  be a set of  $n$  points in the plane, sorted by  $x$ -coordinate. To access the points, we store their  $x$ -coordinates in an  $(a, 2a)$ -tree  $T$  with branching parameter  $a \geq 2$  and leaf parameter  $k \geq 1$ . In particular, every node has degree within  $[a, 2a]$  and every leaf contains at most  $k$  consecutive by  $x$ -coordinate input points. Every internal node  $u$  of  $T$  is associated with an I/O-CPQA whose non-attrited elements correspond to the maximal points among the points stored in the subtree of  $u$ . Moreover,  $u$  contains a *representative block* with the critical records of condition 2 in Lemma 3.1 for the I/O-CPQAs associated with its children nodes.

To construct the structure, we proceed in a bottom up manner. First, we compute the maximal points among the points contained in every leaf of  $T$ . In particular for every leaf, we initialize an I/O-CPQA  $Q$ . We consider the points  $(p_x, p_y)$  stored in the block in increasing  $x$ -coordinate, and call INSERTANDATTRITE( $Q, -p_y$ ). In this way, a point  $p$  in the block that is dominated by another point  $q$  in the block, is inserted before  $q$  in  $Q$  and has value  $-p_y > -q_y$ . Therefore, the dominated points in the block correspond to the attrited elements in  $Q$ .

We construct the I/O-CPQA for an internal node  $u$  of  $T$  by concatenating the already constructed I/O-CPQAs  $Q_i$  at its children nodes  $u_i$  of  $u$ , for  $i \in [1, a]$  in Section 3. Then we call BIAS to the resulting I/O-CPQA appropriately many times in order to satisfy condition 1 in Lemma 3.1. The procedure ends when the I/O-CPQA is constructed for the root of  $T$ . Notice that the order of concatenations follows implicitly the structure of the tree  $T$ . To insert (resp. delete) a point  $p = (p_x, p_y)$  to the structure, we first insert (resp. delete)  $p_x$  to  $T$ . This identifies the leaf with the I/O-CPQA that contains  $p$ . We discard all I/O-CPQAs from the leaf to the root of  $T$ , and recompute them in a bottom up manner, as described above.

To report the skyline among the points that lie within a given 3-sided query rectangle  $[x_\ell, x_r] \times [y_b, +\infty)$ , it is necessary to obtain the maximal points in a subtree of a node  $u$  of  $T$  by querying the I/O-CPQA stored in  $u$ . Notice, however, that computing the I/O-CPQA of an internal node of  $T$  modifies the I/O-CPQAs of its children nodes. Therefore, we can only report the skyline of all points stored in  $T$ , by calling DELETEMIN at the I/O-CPQA stored in the root of  $T$ . The rest of the I/O-CPQAs in  $T$  are not queriable in this way, since the corresponding nodes do not contain the version of their I/O-CPQA, before it is modified by the construction of the I/O-CPQA for their parent nodes. For this reason we render the involved I/O-CPQAs confluent persistent, by implementing their clean, buffer and dirty dequeues as purely functional catenable dequeues [11]. In fact,  $T$  encodes implicitly the directed acyclic version graph of the confluent persistent I/O-CPQAs, by associating every node of  $T$  with the version of the I/O-CPQA at the time of its construction. Every internal node of  $T$  stores a representative block with the critical records for the versions of the I/O-CPQAs associated with its children nodes. Finally, the update operation discards the I/O-CPQA of a node in  $T$ , by performing in reverse the operations on the purely functional catenable dequeues involved in the construction of the I/O-CPQA (undo operation).

With the above modification it suffices for the query operation to identify the two paths  $p_\ell, p_r$  from the root to the leaves of  $T$  that contain the  $x$ -successor point of  $x_\ell$  and the  $x$ -predecessor point of  $x_r$ , respectively. Let  $R$  be the children nodes of the nodes on the paths  $p_\ell$  and  $p_r$  that do not belong to the paths themselves, and also lie within the query  $x$ -range. The subtrees of  $R$  divide the query  $x$ -range into disjoint  $x$ -ranges. We consider the nodes of  $R$  from left to right. In particular, for every non-leaf node in  $p_\ell \cup p_r$ , we load into memory the representative blocks of the versions of the I/O-CPQAs in its children nodes that belong to  $R$ . We call CATENATEANDATTRITE on the loaded I/O-CPQAs and on the resulting I/O-CPQAs for every node in  $p_\ell \cup p_r$ , as described in Section 3. The non-attributed elements in the resulting auxiliary I/O-CPQA correspond to the skyline of the points in the query  $x$ -range, that are not stored in the leaves of  $p_\ell$  and  $p_r$ . To report the output points of the query in increasing  $x$ -coordinate, we first report the maximal points within the query range among the points stored in the leaf of  $p_\ell$ . Then we call DELETEMIN to the auxiliary I/O-CPQA that returns the maximal points in increasing  $x$ -coordinate, and thus also in decreasing  $y$ -coordinate, and thus we terminate the reporting as soon as a skyline point with  $y$ -coordinate smaller than  $y_b$  is returned. If the reporting has not terminated, we also report the rest of the maximal points within the query range that are contained in the leaf of  $p_r$ .

**Theorem 4.1.** *There exist I/O-efficient dynamic data structures that store a set of  $n$  planar points and support reporting the  $t$  skyline points within a given 3-sided orthogonal range unbounded by the positive  $y$ -dimension in  $\mathcal{O}(\log_{2B^\epsilon} n + t/B^{1-\epsilon})$  worst case I/Os, and updates in  $\mathcal{O}(\log_{2B^\epsilon} n)$  worst case I/Os, using  $\mathcal{O}(n/B^{1-\epsilon})$  disk blocks, for a parameter  $0 \leq \epsilon \leq 1$ .*

*Proof.* We set the buffer size parameter  $b$  of the I/O-CPQAs equal to the leaf parameter  $k$  of  $T$ , and we set the parameters  $a = 2B^\epsilon$  and  $k = B^{1-\epsilon}$  for  $0 \leq \epsilon \leq 1$ . In this way, for a node of  $T$ , the representative blocks for all of its children nodes can be loaded into memory in  $\mathcal{O}(1)$  I/Os. Since every operation supported by an I/O-CPQA involves a  $\mathcal{O}(1)$  number of deque operations, I/O-CPQAs can be made confluent persistent without deteriorating their I/O and space complexity. Moreover, the undo operation takes  $\mathcal{O}(1)$  worst case I/Os, since the purely functional catenable dequeues are worst case efficient.

Therefore by Theorem 3.1, an update operation takes  $\mathcal{O}(\log_{2B^\epsilon} \frac{n}{B^{1-\epsilon}}) = \mathcal{O}(\log_{2B^\epsilon} n)$  worst case I/Os. Lemma 3.1 takes  $\mathcal{O}(1)$  I/Os to construct the temporary I/O-CPQAs for every node in the search paths, since they satisfy both of its conditions. Moreover, by Theorem 3.1, it takes  $\mathcal{O}(\frac{\log_{2B^\epsilon} n}{B^{1-\epsilon}})$  I/Os to concatenate them together. Thus, the construction of the auxiliary query I/O-CPQA takes  $\mathcal{O}(\log_{2B^\epsilon} n)$  worst case I/Os

in total. Moreover, it takes  $\mathcal{O}(1 + t/B^{1-\epsilon})$  worst case I/Os to report the output points. There are  $\mathcal{O}(\frac{n}{B^{1-\epsilon}})$  internal nodes in  $T$ , and every internal node contains  $\mathcal{O}(1)$  blocks.  $\square$

**4-Sided Skyline Reporting** Dynamic I/O-efficient data structures for 4-sided range skyline reporting queries can be obtained by following the approach of Overmars and Wood for dynamic rectangular visibility queries [15]. In particular, 4-sided range skyline reporting queries are supported in  $\mathcal{O}(\frac{a \log^2 n}{\log a \log 2B^\epsilon} + t/B^{1-\epsilon})$  worst case I/Os, using  $\mathcal{O}(\frac{n}{B^{1-\epsilon}} \log_a n)$  blocks, by employing our structure for 3-sided range skyline reporting as a secondary structure on a dynamic range tree with branching parameter  $a$ , built over the  $y$ -dimension. Updates are supported in  $\mathcal{O}(\frac{\log^2 n}{\log a \log 2B^\epsilon})$  worst case I/Os, since the secondary structures can be split or merged in  $\mathcal{O}(\log_{2B^\epsilon} n)$  worst case I/Os.

**Remark 4.1.** *In the pointer machine, the above constructions attains the same complexities as the existing structures for dynamic 3-sided and 4-sided range maxima reporting [4], by setting the buffer size, branching and leaf parameter to  $\mathcal{O}(1)$ .*

## 5 Lower Bound for Dominating Minima Reporting

Let  $S$  be a set of  $n$  points in  $\mathbb{R}^2$ . Let  $\mathcal{Q} = \{Q_i\}$  be a set of  $m$  orthogonal 2-sided query ranges  $Q_i \in \mathbb{R}^2$ . Range  $Q_i$  is the subspace of  $\mathbb{R}^2$  that dominates a given point  $q_i \in \mathbb{R}^2$  in the positive  $x$ - and  $y$ - direction (the “upper-right” quadrant defined by  $q_i$ ). Let  $S_i = S \cap Q_i$  be the set of all points in  $S$  that lie in the range  $Q_i$ . A *dominating minima reporting query*  $Q_i$  contains the points  $\min(S_i) \in S_i$  that do not dominate any other point in  $S_i$ . In this section we prove that any pointer-based data structure that supports dominating minima queries in  $\mathcal{O}(\log^{\mathcal{O}(1)} n + t)$  time, must use superlinear space. This separates the problem from the easier problem of supporting dominating maxima queries and the more general 3-sided range skyline reporting queries. The same trade-off also holds for the symmetric *dominated maxima reporting queries* that are the simplest special case of 4-sided range skyline reporting queries that demands superlinear space. Moreover, the lower bound holds trivially for the I/O model, if no address arithmetic is being used. In particular, for a query time of  $\mathcal{O}(\frac{\log^{\mathcal{O}(1)} n}{B} + \frac{t}{B})$  the data structure must definitely use  $\Omega(\frac{n}{B} \frac{\log n}{\log \log n})$  blocks of space. In the following, we prove the lower bound for the dominating minima reporting queries.

Henceforth, we use the terminology presented in Section 2. Without loss of generality, we assume that  $n = \omega^\lambda$ , since this restriction generates a countably infinite number of inputs and thus the lower bound is general. In our case,  $\omega = \log^\gamma n$  holds for some  $\gamma > 0$ ,  $m = 2$  and  $\lambda = \lfloor \frac{\log n}{1 + \gamma \log \log n} \rfloor$ . Let  $\rho_\omega(i)$  be the integer obtained by writing  $0 \leq i < n$  using  $\lambda$  digits in base  $\omega$ , by first reversing the digits and then taking their complement with respect to  $\omega$ . In particular, if  $i = i_0^{(\omega)} i_1^{(\omega)} \dots i_{\lambda-1}^{(\omega)}$  holds, then

$$\rho_\omega(i) = (\omega - i_{\lambda-1}^{(\omega)} - 1)(\omega - i_{\lambda-2}^{(\omega)} - 1) \dots (\omega - i_1^{(\omega)} - 1)(\omega - i_0^{(\omega)} - 1)$$

where  $i_j^{(\omega)}$  is the  $j$ -th digit of number  $i$  in base  $\omega$ . We define the points of  $S$  to be the set  $\{(i, \rho_\omega(i)) | 0 \leq i < n\}$ . Figure 3 shows an example with  $\omega = 4$ ,  $\lambda = 2$ .

To define the query set  $\mathcal{Q}$ , we encode the set of points  $\{(i, \rho_\omega(i)) | 0 \leq i < n\}$  in a full trie structure of depth  $\lambda$ . Recall that  $n = \omega^\lambda$ . Notice that the trie structure is implicit and it is used only for presentation purposes. Input points correspond to the leaves of the trie and their  $y$  value is their label at the edges of the trie. Let  $v$  be an internal node at depth  $d$  (namely,  $v$  has  $d$  ancestors), whose prefix  $v_0, v_1, \dots, v_{d-1}$  corresponds to the path from  $v$  to the root  $r$  of the trie. We take all points in its subtree and sort them by  $y$ . From this sorted list we construct groups of size  $\omega$  by always picking each  $\omega^{\lambda-d-1}$ -th element starting from the smallest non-picked element. Each such group corresponds to the output of each query. See Figure 3 for an example. In this case, we say that the query is *associated* to node  $v$ .

A node of with depth  $d$  has  $\frac{n}{\omega^d}$  points in its subtree and thus it defines at most  $\frac{n}{\omega^{d-1}}$  queries. Thus, the

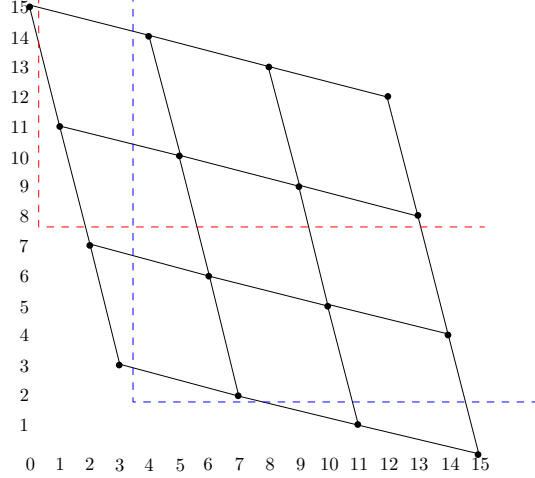


Figure 3: An example for  $\omega = 4$  and  $\lambda = 2$ . Two examples of queries are shown, out of the 8 possible queries with different output. Connecting lines represent points whose  $L_1$  distance is  $\omega^k$ ,  $1 \leq k \leq \lambda$ . All 8 possible queries can be generated by translating the blue lines horizontally so that the answers of all 4 queries are disjoint. Similarly for the red lines with the exception that we translate them vertically.

total number of queries is:

$$|\mathcal{Q}| = \sum_{d=0}^{\lambda-1} \omega^d \frac{n}{\omega^{d+1}} = \sum_{d=0}^{\lambda-1} \frac{n}{\omega} = \frac{\lambda n}{\omega}$$

This means that the total number of queries is

$$|\mathcal{Q}| = \frac{\lambda n}{\omega} = \frac{\log n}{1 + \gamma \log \log n} \frac{1}{\log^\gamma n} n = \frac{n}{\log^{\gamma-1} n (1 + \gamma \log \log n)}$$

The following lemma states that  $\mathcal{Q}$  is appropriate for our purposes.

**Lemma 5.1.**  $\mathcal{Q}$  is  $(2, \log^\gamma n)$ -favorable.

*Proof.* First we prove that we can construct the queries so that they have output size  $\omega = \log^\gamma n$ . Assume that we take a group of  $\omega$  consecutive points in the sorted order of points with respect to the  $y$ -coordinate at the subtree of node  $v$  at depth  $d$ . These have common prefix of length  $d$ . Let the  $y$ -coordinates of these points be  $\rho_\omega(i_1), \rho_\omega(i_2), \dots, \rho_\omega(i_\omega)$  in increasing order, where  $\rho_\omega(i_j) - \rho_\omega(i_{j-1}) = \omega^{\lambda-d-1}$ ,  $1 < j \leq \omega$ . This means that these numbers differ only at the  $\lambda - d - 1$ -th digit. This is because they have a common prefix of length  $d$  since all points lie in the subtree of  $v$ . At the same time they have a common suffix of length  $\lambda - d - 1$  because of the property that  $\rho_\omega(i_j) - \rho_\omega(i_{j-1}) = \omega^{\lambda-d-1}$ ,  $1 < j \leq \omega$  which comes as a result from the way we chose these points. By inverting the procedure to construct these  $y$ -coordinates, the corresponding  $x$ -coordinates  $i_j$ ,  $1 \leq j \leq \omega$  are determined. By complementing we take the increasing sequence  $\bar{\rho}_\omega(i_\omega), \dots, \bar{\rho}_\omega(i_2), \bar{\rho}_\omega(i_1)$ , where  $\bar{\rho}_\omega(i_j) = \omega^\lambda - \rho_\omega(i_j) - 1$  and  $\bar{\rho}_\omega(i_{j-1}) - \bar{\rho}_\omega(i_j) = \omega^{\lambda-d-1}$ ,  $1 < j \leq \omega$ . By reversing the digits we finally get the increasing sequence of  $x$ -coordinates  $i_\omega, \dots, i_2, i_1$ , since the numbers differ at only one digit. Thus, the group of  $\omega$  points are decreasing as the  $x$ -coordinates increase, and as a result a query  $q$  whose horizontal line is just below  $\rho_\omega(i_1)$  and the vertical line just to the left of  $\rho_\omega(i_\omega)$  will certainly contain this set of points in the query. In addition, there cannot be any other points between this sequence and the horizontal or vertical lines defining query  $q$ . This is because all points in the subtree of  $v$  have been sorted with respect to  $y$ , while the horizontal line is positioned just below  $\rho_\omega(i_1)$ , so that no other element lies in between. In the same manner, no points to the left of  $\rho_\omega(i_\omega)$  exist, when positioning the vertical line of  $q$  appropriately. Thus, for each query  $q \in \mathcal{Q}$ , it holds that  $|S \cap q| = \omega = \log^\gamma n$ .

It is enough to prove that for any two query ranges  $p, q \in \mathcal{Q}$ ,  $|S \cap q \cap p| \leq 1$  holds. Assume that  $p$  and  $q$  are associated to nodes  $v$  and  $u$ , respectively, and that their subtrees are disjoint. That is,  $u$  is not a proper ancestor or descendant of  $v$ . In this case,  $p$  and  $q$  share no common point, since each point is used only once in the trie. For the other case, assume without loss of generality that  $u$  is a proper ancestor of  $v$  ( $u \neq v$ ). By the discussion in the previous paragraph, each query contains  $\omega$  numbers that differ at one and only one digit. Since  $u$  is a proper ancestor of  $v$ , the corresponding digits will be different for the queries defined in  $u$  and for the queries defined in  $v$ . This implies that there can be at most one common point between these sequences, since the digit that changes for one query range is always set to a particular value for the other query range. The lemma follows.  $\square$

Lemma 5.1 allows us to apply Lemma 2.1, and thus the query time of  $\mathcal{O}(\log^\gamma n + t)$ , for output size  $t$ , can only be achieved at a space cost of  $\Omega\left(n \frac{\log n}{\log \log n}\right)$ . The following theorem summarizes the result of this section.

**Theorem 5.1.** *The dominating minima reporting problem can be solved with  $\Omega\left(n \frac{\log n}{\log \log n}\right)$  space, if the query is supported in  $\mathcal{O}(\log^\gamma n + t)$  time, where  $t$  is the size of the answer to the query and parameter  $\gamma = \mathcal{O}(1)$ .*

## 6 Conclusion

We presented the first dynamic I/O-efficient data structures for 3-sided planar orthogonal range skyline reporting queries with worst case polylogarithmic update and query complexity. We also showed that the space usage of the existing structures for 4-sided range skyline reporting in pointer machine is optimal within doubly logarithmic factors.

It remains open to devise a dynamic I/O-efficient data structure that supports reporting all  $m$  planar skyline points in  $\mathcal{O}(m/B)$  worst case I/Os and updates in  $\mathcal{O}(\log_B n)$  worst case I/Os. It seems that the hardness for reporting the skyline in optimal time is derived from the fact that the problem is dynamic. The dynamic indexability model of Yi [21] may be useful to prove a lower bound towards the direction of rendering our structure for 3-sided range skyline reporting *I/O-optimal*, as defined by Papadias et al.[16]. Finally it remains open to obtain a  $\mathcal{O}\left(\frac{n}{B} \log_B n\right)$  space dynamic I/O-efficient data structures for 4-sided range skyline reporting with  $\mathcal{O}(\log_B^2 n)$  worst case query and update I/Os, regardless of the I/O-complexity per reported point.

## References

- [1] Alok Aggarwal and S. Vitter, Jeffrey. The input/output complexity of sorting and related problems. *Commun. ACM*, 31(9):1116–1127, 1988.
- [2] Jon Louis Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, April 1980.
- [3] Stephan Börzsönyi, Donald Kossmann, and Konrad Stocker. The skyline operator. In *ICDE*, pages 421–430, 2001.
- [4] Gerth Brodal and Konstantinos Tsakalidis. Dynamic planar range maxima queries. In Luca Aceto, Monika Henzinger, and Jir Sgall, editors, *Automata, Languages and Programming*, volume 6755 of *Lecture Notes in Computer Science*, pages 256–267. Springer Berlin / Heidelberg, 2011. 10.1007/978-3-642-22006-7\_22.
- [5] Bernard Chazelle. Lower bounds for orthogonal range searching: I. the reporting case. *J. ACM*, 37(2):200–212, April 1990.
- [6] Bernard Chazelle and Ding Liu. Lower bounds for intersection searching and fractional cascading in higher dimension. *Journal of Computer and System Sciences*, 68(2):269 – 284, 2004. `jc:title;Special Issue on STOC 2001;ce:title;`
- [7] Ananda Das, Prosenjit Gupta, Anil Kalavagattu, Jatin Agarwal, Kannan Srinathan, and Kishore Kothapalli. Range aggregate maximal points in the plane. In Md. Rahman and Shin-ichi Nakano, editors, *WALCOM: Algorithms and Computation*, volume 7157 of *Lecture Notes in Computer Science*, pages 52–63. Springer Berlin / Heidelberg, 2012. 10.1007/978-3-642-28076-4\_8.
- [8] Yu-Ling Hsueh, Roger Zimmermann, and Wei-Shinn Ku. Efficient updates for continuous skyline computations. In *DEXA*, pages 419–433, 2008.
- [9] Zhiyong Huang, Hua Lu, Beng Chin Ooi, and Anthony K. H. Tung. Continuous skyline queries for moving objects. *IEEE Trans. Knowl. Data Eng.*, 18(12):1645–1658, 2006.
- [10] Anil Kishore Kalavagattu, Ananda Swarup Das, Kishore Kothapalli, and Kannan Srinathan. On finding skyline points for range queries in plane. In *CCCG*, 2011.
- [11] Haim Kaplan and Robert E. Tarjan. Purely functional, real-time deques with catenation. *J. ACM*, 46(5):577–603, September 1999.
- [12] 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.
- [13] Michael D. Morse, Jignesh M. Patel, and William I. Grosky. Efficient continuous skyline computation. *Inf. Sci.*, 177(17):3411–3437, 2007.
- [14] Mark H. Overmars and Jan van Leeuwen. Maintenance of configurations in the plane. *Journal of Computer and System Sciences*, 23(2):166 – 204, 1981.
- [15] Mark H. Overmars and Derick Wood. On rectangular visibility. *J. Algorithms*, 9(3):372–390, September 1988.
- [16] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. Progressive skyline computation in database systems. *ACM Trans. Database Syst.*, 30(1):41–82, 2005.
- [17] Cheng Sheng and Yufei Tao. On finding skylines in external memory. In *Proceedings of the thirtieth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '11, pages 107–116, New York, NY, USA, 2011. ACM.

- [18] Rajamani Sundar. Worst-case data structures for the priority queue with attrition. *Inf. Process. Lett.*, 31:69–75, April 1989.
- [19] Yufei Tao and Dimitris Papadias. Maintaining sliding window skylines on data streams. *IEEE Trans. on Knowl. and Data Eng.*, 18(3):377–391, 2006.
- [20] Ping Wu, Divyakant Agrawal, Ömer Egecioglu, and Amr El Abbadi. Deltasky: Optimal maintenance of skyline deletions without exclusive dominance region generation. In *ICDE*, pages 486–495, 2007.
- [21] Ke Yi. Dynamic indexability and lower bounds for dynamic one-dimensional range query indexes. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '09, pages 187–196, New York, NY, USA, 2009. ACM.