Dynamically Maintaining the Persistent Homology of Time Series

July 13, 2023

Abstract

We present a dynamic data structure for maintaining the persistent homology of a time series of real numbers. The data structure supports local operations, including the insertion and deletion of an item and the cutting and concatenating of lists, each in time $O(\log n + k)$, in which n counts the critical items and k the changes in the augmented persistence diagram. To achieve this, we design a tailor-made tree structure with an unconventional representation, referred to as banana tree, which may be useful in its own right.

1 Introduction

Persistent homology is an algebraic method aimed at the topological analysis of data; see e.g. [3, 9]. It applies to low-dimensional geometric as well as to high-dimensional abstract data. In a nutshell, the method is the embodiment of the idea that features exist on many scale levels, and rather than preferring one scale over another, it quantifies the features in terms of the range of scales during which they appear. More precisely, the goal of persistent homology is to compute a representation of the features in a book-keeping data structure, called *persistence* or *persistence diagram* [9].

Importantly, persistent homology has fast algorithms that support the application to large data sets. Specifically, for arbitrary dimensional inputs, persistence is generally computed by analyzing an underlying complex, with worst-case time cubic in the size of the complex. These algorithms are, however, observed to run much faster in applications; see e.g. [14] for a survey on popular implementations. We restrict ourselves to one-dimensional input data, i.e., a list of m points (or *items*) in an interval of $\mathbb{R}$ with each item i being assigned a *value* $f(i)$. Examples that use one-dimensional persistent homology are time series such as heart-rate data [4, 12], gene expression data [7, 15], and financial data [10].

The persistent homology of one-dimensional data can be derived from the merge tree [17], which records more detailed information about the structure of the persistence diagram, called the *history of the connected components in the filtration of sublevel sets.* Without recovering this history, the persistence information can be computed in $O(m)$ time [11]. To the best of our knowledge, the new $O(m)$ time algorithm in this paper is the first linear-time algorithm that can also recover the history, which we store in the *augmented persistence diagram.* The augmented persistence diagram (more formally, the augmented persistence diagram of the sublevel set filtration) is the extended persistence diagram of [5] together with a relation that encodes the merge tree. It is defined formally in Section 2.3.

As the data may change, it is an interesting question whether persistent homology can be maintained efficiently under update operations. The historically first such algorithm [6] takes time linear in the complex size per swap in the ordering of the simplices; see also [13, 16]. For one-dimensional data this corresponds to a change of the value of an item, which reduces to a sequence of interchanges of f-values, each costing time linear in the size of the persistence diagram. Furthermore, with a severe penalty in running-time, this algorithm was applied to a more general version of persistence, called zigzag persistence [8]. The current paper is the first to maintain the persistent homology of dynamically changing one-dimensional input with a tailor-made data structure under a large set of update operations. The running time per operation of our data structure is logarithmic in the number of critical items plus linear in the change of the output.

Our novel dynamic data structure is based on the characterization of the items in the persistence diagram through *windows*, as recently established in [2]. The main data structure is a *binary tree* ordered by position as well as value (see [18] for the introduction of such a binary tree), a *path-decomposition* of this tree dictated by persistent homology such that each path represents a window, and a final relaxation obtained by splitting each path into a *left trail* of nodes with right children on the path and a *right trail* of nodes with left children on the path. Our algorithms perform various local operations in time $O(\log n + k)$, in which n is the current number of critical items, and k is the number of changes to the augmented persistence diagram caused by the operation. The suite of local operations includes the *insertion* of a new item, the *deletion* of an item, the *adjustment* of the value of an item, the *cutting* of a list of items into two, and the *concatenation* of two lists into one. The split of each path into two trails is crucial for these results, and without it, the update time would have a linear dependence on the depth of the tree, which might be $\Theta(n)$.

Outline. Section 2 provides the background needed for this paper: lists and maps, persistent homology, and the hierarchy of windows that characterizes the augmented persistent diagram. In Section 3 we present a technical overview of our technique. Section 4 introduces the data structures we use to represent a linear list: a doubly-linked list, two dictionaries, and two path-decomposed ordered binary trees whose paths are stored as pairs of trails. Section 5 presents the algorithms for maintaining the augmented persistence diagram of a linear list. Section 6 concludes the paper.

2 Background

This section explains how a linear list can be viewed as a continuous map on a closed interval. Looking at the sub- and superlevel sets of such a map, we define its augmented persistence diagram and review the hierarchical characterization in terms of windows, as proved in [2]. A modest amount of topology suffices to describe these diagrams, and we refer to [9] for a more comprehensive treatment needed to place the results of this paper within a larger context.

2.1 Lists Viewed as Maps

By a *linear list* we mean a finite sequence of real numbers, $c_1, c_2, \ldots, c_m$. To view the list as a continuous map, we set $f(i) = c_i$, for $1 \leq i \leq m$, and linearly interpolate between consecutive values. The result is a piece-wise linear map on a closed interval, $f\colon [1,m] \to \mathbb{R}$, with *items* i and *values* c_i, for $1 \leq i \leq m$. Important about the items is their ordering and not their precise positions along the interval, so we use consecutive integers for convenience. To simplify discussions, we will often assume that the map is *generic*, by which we mean that its items have distinct values. This is no loss of generality since a small perturbation may be simulated and implemented by appropriate tie-breaking rules.

Given $t \in \mathbb{R}$, the *sublevel set* of f at t, denoted $f_t = f^{-1}(-\infty, t]$, is the set of points in domain of f such that its value $f(x)$ is not larger than any fixed point $t \in \mathbb{R}$. Since f is defined on a closed interval, the homology of f_t is fully characterized by the number of connected components. A point x in the closed interval, $[1,m]$ is a *(homological) critical point* of f if the number of connected components of f_t changes at the moment t passes $f(x)$. Assuming genericity, a critical point is necessarily an item of f, and the only two types in the interior of $[1,m]$ are *minima* and *maxima*. When t passes the value of a minimum from below, then the number of connected components of f_t increases by 1, and if t passes the value of a maximum from below, the number of connected components decreases by 1. The endpoints of $[1,m]$ are special: the number of connected components changes at an *up-type endpoint* and it remains unchanged at a *down-type* endpoint. Symmetrically, we call $f^t = f^{-1}[t, \infty)$ the *superlevel set* of f at t. Observe that f^t is the sublevel set of $-f$ at $-t$, and that the minima and maxima of $-f$ are the maxima and minima of f. Similarly, the endpoints swap type.

2.2 Extended Persistent Homology

Persistent homology tracks the evolution of the connected components while the sublevel set of f grows, and formally defines when a component is born and when it dies. Complementing this with the same information for the superlevel sets of f, we get what is formally referred to as *extended persistent homology*, which we explain next (see [5] for more details).

In *Phase One*, we track the connected components of the sublevel set, f_t, as t increases from $-\infty$ to ∞. A component is *born* at the smallest value of t at which a point of the component belongs to f_t. This point is necessarily a minimum in the interior of $[1,m]$, or an up-type endpoint.

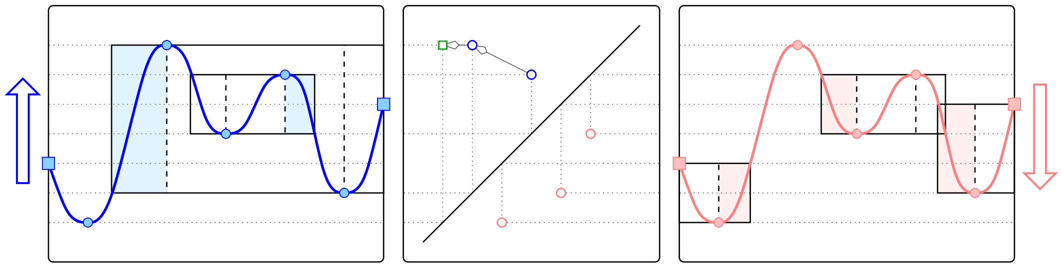

Figure 1: *Left:* a real-valued map on a closed interval, f, with three minima and two maxima. *Right:* the map $-f$ drawn upside-down. *Middle:* the augmented persistence diagram with two (*blue*) points in the ordinary subdiagram $\mathsf{Ord}(f)$ above the diagonal, three (*pink*) points in the relative subdiagram $\mathsf{Rel}(f)$ below the diagonal, and one (*green*) point in the essential subdiagram $\mathsf{Ess}(f)$.

The component *dies* when it merges with another component that was born earlier. The point at which the two components merge is necessarily a maximum in the interior of $[1, m]$. The *ordinary subdiagram* of f, denoted $\mathsf{Ord}(f)$, records the birth and death of every component with a point in the plane whose abscissa and ordinate are those values of t at which the component is born and dies, respectively; see Figure 1.

In *Phase Two*, we track the connected components of the superlevel set, f^t, as t decreases from ∞ to $-\infty$. Birth and death are defined accordingly, and the components are recorded in the *relative subdiagram*, denoted $\mathsf{Rel}(f)$. By construction, the points in $\mathsf{Ord}(f)$ lie above and those of $\mathsf{Rel}(f)$ lie below the diagonal; see Figure 1. The component born at the global minimum of f is special because it does not die during Phase One. Instead, it dies at the global minimum of $-f$, which is the global maximum of f. In topological terms, this happens because the one connected component still alive at the beginning of Phase Two dies in relative homology when its first point enters the superlevel set.[1] This class is represented by the sole point in the *essential subdiagram*, denoted $\mathsf{Ess}(f)$; see again Figure 1. The *extended persistence diagram*, denoted $\mathsf{Dgm}(f)$, is the disjoint union of the three subdiagrams. Hence, $\mathsf{Dgm}(f)$ is a multi-set of points in $\mathbb{R}^2$, and so are the three subdiagrams, unless the map is generic, in which case the diagram is a set.

2.3 Windows and Augmented Persistence Diagram

The points of the persistence diagram can be characterized using the concept of windows recently introduced in [2]. Let a and b be two (homological) critical points of f, with values $A = f(a) < f(b) = B$. Note that $f^{-1}[A, B]$ contains all points in the interval whose values lie between A and B. It consists of one or more connected components, and it is quite possible that a and b belong to different components. However, if they belong to the same component, then we denote by $[x, y]$ the connected component of $f^{-1}[A, B]$ that contains both a and b and we call $[x, y] \times [A, B]$ the *frame* with *support* $[x, y]$ spanned by a and b. There are two orientations of the frame: from left to right if $a < b$, and from right to left if $b < a$. In the former case, we call x the *mirror* of b and $W(a, b)$ a *triple-panel window* if $f(y) = A$. In the latter case, we call y the *mirror* of b and $W(a, b)$ a *triple-panel window* if $f(x) = A$. We say a and b *span* $W(a, b)$. We distinguish a special type

[1]In the original formulation in [5], Phase Two in the construction of the (extended) persistence diagram tracks the 1-dimensional relative homology groups of the pairs (f_∞, f^t), which are isomorphic to 0-dimensional homology groups of the f^t. To appreciate the guiding hand of algebraic topology, we need to understand how the absolute and relative homology groups of different dimensions relate to each other. However, for the purpose of this paper, this is not necessary and we can track the connected components of the superlevel set in lieu of the relative cycles in the interval modulo the superlevel set.

of window, referred to as a *global window*, whose support covers the entire interval, $[x, y] = [1, m]$, and which is exempt of the requirement at x or y.

In [2], the authors prove that there is a bijection between the windows and the points in $\mathsf{Dgm}(f)$. Furthermore, each critical point in the interior of the interval spans exactly one window in each phase—even though the pairing may be different in the two phases—while each endpoint spans a window in only one phase. In Figure 1, there are five triple-panel windows, two on the left and three on the right. There is always exactly one global window spanned by the global minimum, α, and the global maximum, β, of f. What follows is a special case of a more general characterization of persistence proved in [2].

Proposition 2.1 (Persistence in Terms of Windows [2])**.** *Let $f\colon [1, m] \to \mathbb{R}$ be a generic piecewise linear map on a closed interval, and let a, b be homological critical points of f, or of $-f$, with $f(a) = A$ and $f(b) = B$. Then*

(i) $(A, B) \in \mathsf{Ord}(f)$ *iff $W(a, b)$ is a triple-panel window of f,*

(ii) $(B, A) \in \mathsf{Rel}(f)$ *iff $W(b, a)$ is a triple-panel window of $-f$,*

(iii) $(A, B) \in \mathsf{Ess}(f)$ *iff $W(a, b)$ is the global window of f.*

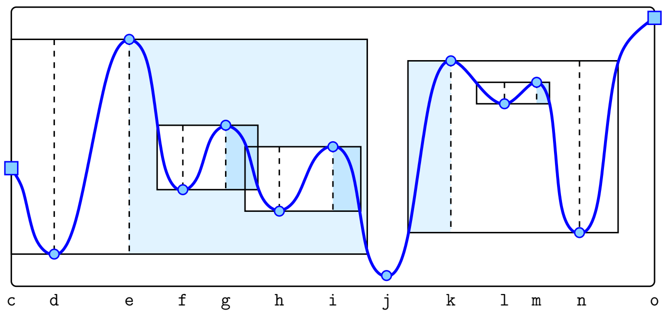

Figure 2: The graph of a generic map on a closed interval. All windows shown are with simple wave, except for the *leftmost* window, whose wave is short. The global window as well as the (tiny) windows caused by the hooks (which will be introduced in section 4) are not shown. The *light-blue shaded* out-panels are part of the triple- but not of the double-panel windows.

Suppose $W(a, b)$ is a triple-panel window, with support $[x, y]$. Unless a is an endpoint of $[1, m]$, cutting $[x, y]$ at a and b produces three segments, $[x, a]$, $[a, b]$, $[b, y]$, in which we assume $a < b$. We call the corresponding products with $[A, B]$ the *in-panel*, *mid-panel*, *out-panel*, and the graph within the window a *wave*. This wave is *simple* or *short* depending on whether the function value at the mirror is equal to or smaller that at the maximum: $f(x) = B$ or $f(x) < B$. In Figure 2, we have four simple waves (defined by $\mathtt{f}, \mathtt{g}$, by $\mathtt{h}, \mathtt{i}$, by $\mathtt{l}, \mathtt{m}$, and by $\mathtt{n}, \mathtt{k}$) and one short wave (defined by $\mathtt{d}, \mathtt{e}$). Note that a simple wave of f is also a simple wave of $-f$, albeit upside-down. A similar statement does not hold for short waves, which explains the violations of symmetry in the persistence diagram; see again Figure 1.

The *double-panel* version of a triple-panel window consists of the in-panel and the mid-panel but drops the out-panel. Any two double-panel windows of f are either disjoint or nested, and all these windows are nested inside the global window [2, Lemma 3.3]. We use this property to augment the persistence diagram by drawing an arrow between two points if the corresponding double-panel windows are nested without any other window nested between them. The result is the *augmented persistence diagram*, denoted $\overrightarrow{\mathsf{Dgm}}(f)$.

In this paper, we consider operations that maintain the augmented persistence diagram to reflect the change from a map, f, to any other map, g. We quantify this difference by the number of points and arrows that change or, more formally, belong to the symmetric difference between the two diagrams. What are typical differences? Every local change in the function reduces to a sequence of *interchanges* between two minima or two maxima, possibly ending in a *cancellation* or beginning with an *anti-cancellation*. Every cancellation removes a point from the diagram, and every anti-cancellation adds a point to the diagram, so the diagrams before and after differ by a constant amount. An interchange may or may not swap two critical values between points (two coordinates, which are values of two critical items) or the reversal of an arrow, and if it does not, then this should not incur any cost. On the other hand, if it does, then this can cost a constant amount of time. Our declared goal is to have operations that run in time $O(\log n + k)$, in which n is the number of critical points and k is the size of the symmetric difference between $\overrightarrow{\mathsf{Dgm}}(f)$ and $\overrightarrow{\mathsf{Dgm}}(g)$.

3 Technical Overview

To achieve this goal, we propose a novel collection of data structures—some classic and some new—for maintaining nested windows. We show that using these data structures allows us to maintain the augmented persistence diagram to reflect the change from one map to another in the desired time bound. We maintain the following data structures:

(1) a *doubly-linked list* of all items, critical or not, ordered by their positions in the interval;

(2) two *binary search trees*, called *dictionaries*, one storing the minima and the other storing the maxima, both ordered by their positions in the interval;

(3) two *banana trees* (described next) representing the information in the augmented persistence diagram by storing the minima and maxima while reflecting their ordering by position as well as by function value.

Note that the minima and maxima are subsets of all items and are therefore represented in all of the above data structures. To reduce the special cases in the algorithms, we add two artificial items, called *hooks*, at the very beginning and the very end of the interval. They make sure that the formerly first and last items are proper minima or maxima, but we ignore them and the technicalities involved in this overview and give slightly simplified descriptions of the data structures and the algorithms.

Banana Trees. We introduce these trees in three stages. In the *first stage*, we organize all windows in a full binary tree, whose leaves are the minima and whose internal nodes are the maxima, such that (i) the in-order traversal of the tree visits the nodes in increasing order of their positions in the interval, and (ii) the nodes along any path from a leaf to the root are ordered by increasing function value. Such a tree always exists and it is unique. In the following, we do not distinguish between a node in the tree and the critical item it represents.

In the *second stage*, we add a *special root* labeled with value larger than the global maximum whose only child is the previous root. Call the resulting tree T and note that it has an equal number of leaves and internal nodes. We then form paths, each starting at a leaf, a, and ending at an internal node, b, such that a and b span a window, $W(a, b)$. Call this path $P(a, b)$. Based on

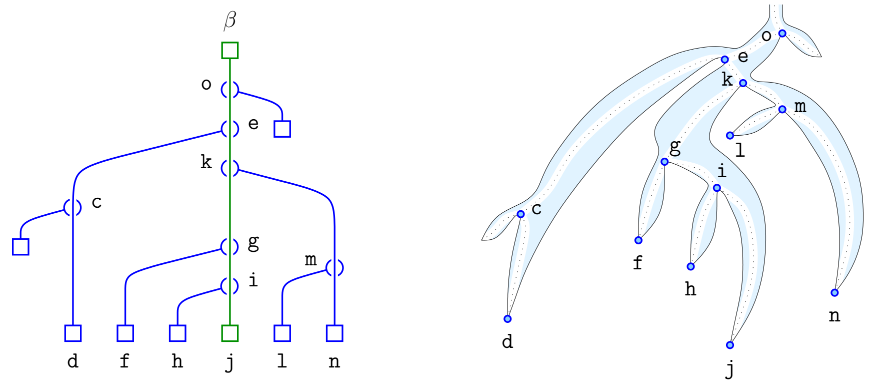

Figure 3: The path-decomposed binary tree associated to the map in Figure 2 with special root, β, on the *left*, and the corresponding banana tree on the *right*.

structural properties of T, this leads to a partition of T into edge-disjoint (but not vertex-disjoint) paths; see the left drawing in Figure 3. The node b ending the path that starts at a is locally determined: it is the first internal node encountered while walking up from a, such that a and the descending leaf with minimum function value lie on different sides (in different subtrees) of this internal node. Hence, every maximum on the path from a to b spans a window that is immediately nested in $W(a,b)$. It follows that every maximum, b, belongs to two paths: $P(a,b)$ and $P(p,q)$ such that $W(a,b)$ is immediately nested in $W(p,q)$. This even holds for the root (of the full binary tree), which spans a path and also belongs to the path that ends at the special root. Given a map, f, the tree, T, and its partition into paths are unique. The strict dependence on f may force T to be unbalanced, and indeed have linear depth, so that efficient maintenance algorithms are challenging. This is why we need another modification.

In the *third stage*, we split each path into two trails; see the right drawing in Figure 3. The *left trail* of $P(a,b)$ contains a and every maximum u on $P(a,b)$ with $u \leq a$, while the *right trail* contains a and every maximum u on $P(a,b)$ with $a \leq u$. A node v on $P(a,b)$ is the right child of its parent, u, in the binary tree iff u is on the left trail. Furthermore, v is the left child of u iff u is on the right trail. Thus, to which trail u belongs to can be decided based on local information at u. To simplify language, we also give a second name to the trails. If $a < b$, then $W(a,b)$ consists of the in-panel on the left, the mid-panel in the middle, and the out-panel on the right. In this case, u belongs to the left trail iff the window it spans is nested inside the in-panel, so we alternatively call the left trail the *in-trail*, and we alternatively call the right trail the *mid-trail*. If $b < a$, then the right trail is the in-trail and the left trail is the mid-trail. So what happened to the out-trail? It is indeed needed, but only if $W(a,b)$ is with simple wave. Such a window corresponds to $P(a,b)$ in the banana tree of f and to $P(b,a)$ in the banana tree of $-f$. The out-panel of $W(a,b)$ is the in-panel of $W(b,a)$, so we can get information about the out-panel from the in-panel of the same window in the other banana tree when we need it.

To speed up the algorithms, we maintain various pointers, such as between the occurrences of the same critical item in the banana trees, the dictionaries, and the doubly-linked list, for example. Importantly, every internal node, b, stores a pointer to the descending leaf with minimum function value, $\mathsf{low}(b)$, and every leaf, a, stores a pointer to the endpoint of its path, $\mathsf{dth}(a)$. Observe that $q = \mathsf{dth}(\mathsf{low}(b))$ is the maximum that spans the window in which $W(a,b)$ is immediately nested, so this window can be obtained in constant time.

Construction. While the main focus of this paper is the maintenance of the data structures through local updates, we also consider the construction from scratch. It is straightforward to derive the augmented persistence diagram during a single traversal of the banana trees in linear time, so the question we study is how fast this diagram can be constructed from a given one-dimensional input list. Assuming all non-critical items have been removed and we are given the remaining sequence of n critical items, there are standard algorithms that can be adapted to construct the banana trees in $O(n \log n)$ time. There is also an $O(n)$ time algorithm for computing the persistence diagram [11], but this algorithm does not extend to the augmented persistence diagram. To the best of our knowledge our algorithm is the first to construct the augmented persistence diagram in $O(n)$ time.

The main structure of the algorithm is a left-to-right scan of the data. We interpret the item i with value $c_i = f(i)$ as the point (i, c_i) in the plane and maintain a decreasing staircase such that all processed items are points on or below the staircase. Each step of the staircase corresponds to an unfinished banana. One of the difficulties is that before a banana is completed, we do not know whether it will be attached to a left or a right trail. We tentatively assume it will be attached to a left trail but are prepared to move the banana to the other side when this turns out to be necessary. When we process the next item, we may remove any number of steps, turning each into a finished banana, but we can add at most one new step. Since a step that is removed was added earlier, this proves that the algorithm runs in $O(1)$ amortized time per item, and therefore in $O(n)$ time altogether.

Local Maintenance. Given a list of m items with real function values, we consider the operations that *insert* a new item, *delete* an item, and *change the value* of an item. All three operations reduce to a sequence of *interchanges*—which can be between two maxima or between two minima—possibly preceded by an *anti-cancellation* or a *slide*, and possibly succeeded by a *cancellation* or a *slide*. In a slide, a minimum or maximum next to a non-critical item becomes non-critical, and the non-critical item becomes a minimum or maximum, respectively. Similarly in a cancellation, a minimum and a neighboring maximum simultaneously become non-critical. Here we will focus on the interchanges, because they are most common as well as most interesting, and on the anti-cancellations, because they pose an unexpected challenge.

Consider two maxima, b and q, of f, and assume $f(b) < f(q)$ before the interchange. To avoid confusing language, we write $g(b)$ and $g(q)$ for the values after the operation but assume that f and g agree on all items except for b. Furthermore, we assume that $g(b) > g(q)$ and that b and q are the only two items for which the ordering by f-value differs from the ordering by g-value. In many cases, the interchange of b and q does not affect the banana trees. Indeed, only if b is a child of q is it necessary to update the order of the two nodes. And even if b and q are consecutive maxima on a path, there is no structural change unless b and q also belong to a common trail. This is the main reason for splitting each path into two trails as explained in the third stage of the introduction of the banana trees: to avoid any cost to occur for interchanges that have no structural affect on the augmented persistence diagram. When b and q interchange while belonging to different trails, then the banana tree is oblivious to this change and requires no update. On the other hand, if b and q belong to the same trail, then they swap positions along this trail, and there is a change of the augmented persistence diagram that pays for the time it takes to update the banana tree.

The interchange of two minima, a and p, is quite different because it does not affect the ordered binary tree at the first stage of the banana tree. However, the interchange affects the path-decomposition, so some of the bananas may have to be updated. To be specific, assume $f(a) > f(p)$ before and $g(a) < g(p)$ after the operation. As before, we also assume that f and g agree on all items except for a, and that a and p are the only two items for which the ordering by f-value differs from the ordering by g-value. Let b and q be the internal nodes so that $P(a, b)$ and $P(p, q)$ are paths in the decomposition of the banana tree of f. The interchange of a and p has no effect on the banana tree, unless b is a node on $P(p,q)$. If b lies on $P(p,q)$, then we extend $P(a,b)$ to $P(a,q)$ and we shorten $P(p,q)$ to $P(p,b)$. The nodes u on the path from b to the child of q change their pointer from $\mathsf{low}(u) = p$ to $\mathsf{low}(u) = a$. There can be arbitrarily many such nodes, but each change causes the adjustment of an arrow in the extended persistence diagram, to which it can be charged in the running time analysis.

This shows that it is possible to perform an interchange of two minima within the desired time bound, but it is not clear how to find them. Considering the scenario in which a decreases its value continuously, it may cause a sequence of interchanges with other minima, but since these minima are not sorted by function value, it is not clear how to find them, and how to ignore the ones without structural consequences. Here is where the relation between the banana trees of f and $-f$ becomes important. The minima of f are the maxima of $-f$, so the interchange of two minima in the banana tree of f corresponds to the interchange of two maxima in the banana tree of $-f$. We already know how to find the interchanges of two maxima and how to ignore the ones without structural consequences, so we use them to identify the interchanges of minima. More precisely, we prove that the interchange of the minima, a and p of f, affects the structure of the banana tree of f only if the interchange of the maxima, a and p of $-f$, affect the structure of the banana tree of $-f$. The converse does not hold, but the implication suffices since the interchange of the maxima of $-f$ leads to a change in the extended persistence diagram which can be charged for the interchange of the minima, which costs only $O(1)$ time if there are no structural adjustments.

Next, we sketch what happens in the remaining operations. A *slide* occurs if a maximum decreases its value so that it becomes non-critical, while a neighboring non-critical item becomes a maximum. However, if this neighboring item is a maximum, then both items become non-critical at the same time, in which case the operation is called a *cancellation*. There are also the symmetric operations in which a minimum increases its value and becomes non-critical, while a neighboring item changes from non-critical to minimum (a *slide*) or from maximum to non-critical (a *cancellation*). The corresponding updates are easily performed within the required time bounds.

A more delicate operation is the *anti-cancellation*, in which two neighboring non-critical items become critical at the same moment in time. Let a and b be these two items and assume a is a minimum and b is a maximum after the operation, so $g(a) < g(b)$ and, by assumption, $f(a') > f(a) > f(b) > f(b')$, in which a', a, b, b' are four consecutive items in the list. Hence, a and b are not present in the banana tree of f, but $P(a,b)$ is a path in the path-decomposed tree of g, so a, b form a minimal banana in the banana tree of g. All we need to do is find the correct place to attach this banana, but this turns out to be difficult. We first explain why it is difficult, then present an algorithm that works within the current data structure, and finally sketch a modification of the data structure that accelerates the anti-cancellation to $O(\log n)$ time. While this leads to a speed-up in the worst case for this type of operation, it slows down other operations by a logarithmic factor.

We use mirrors to explain in what situation an anti-cancellation is difficult. Their representation in the banana tree is indirect: the maximum is the upper end of a mid-trail, and its mirror is the upper end of the matching in-trail. In the tree, the two upper ends are the same node, but if we traverse the trails of the banana trees in sequence, we visit them at different times, and these times correspond to the positions along the interval, which are different for the maximum and its mirror. Every maximum has at most one mirror, so adding all mirrors to the list increases it to less than double the original size. Nevertheless, it is easy to construct a case in which there is a long subsequence of mirrors, and these mirrors are consecutive with the exception of a and b appearing somewhere in their midst. In this situation, finding the correct attachment of $P(a,b)$ in the banana tree of g needs the mirrors immediately to the left and right of b. The data structure as described provides no fast search mechanisms among the mirrors, so we find the correct attachment by linear search starting from the first maximum to the left of a. Suppose we pass k mirrors before we find this attachment. Then we have a nested sequence of k windows, and $W(a, b)$ is nested inside all of them. Hence, there is one new immediate nesting pair, while the transitive closure of the nesting relation gains k new pairs. The cost of the anti-cancellation can be charged to the change of this transitive closure. In many cases, the change in the transitive closure will be comparable in size to that of the transitive reduction, but it can also be significantly larger, like in the case we just described. We thus entertain alternatives.

There is a modification of the data structure that allows for fast searches among subsequences of mirrors: for any consecutive min-max pair in the list, store the mirrors between them in a binary search tree. In practice, there will be many very small such trees, but it is possible that the mirrors accumulate and produce a few large trees. With this modification, an anti-cancellation can be performed in $O(\log n)$ time. Note however, that these binary search trees have to be maintained, which adds a factor of $O(\log n)$ to the time of every operation, in particular to every interchange, of which there can be many.

Topological Maintenance. We call an operation *topological* if it cuts the list of points into two, or it concatenates two lists into one. More challenging topological operations, such as gluing the two ends of a list to form a cyclic list, or gluing several lists to form a geometric tree or more complicated geometric network are feasible but beyond the scope of this paper.

When we *cut* the list of data into two, we *split* the banana trees of the map and its negative into two each. We mention both banana trees since we need information from the other to be able to split one within the desired time bound. Splitting one banana tree is superficially similar to splitting a binary search tree, but more involved. Let $f\colon [1, m] \to \mathbb{R}$ be the map before the operation and $g\colon [0, \ell] \to \mathbb{R}$ and $h\colon [\ell+1, m] \to \mathbb{R}$ the maps after the operation. We write $z = \ell + \frac{1}{2}$ for the position at which the time series is cut. Since the banana trees are determined by the maps, we need to understand the difference between the windows of f and those of g and h. Whether or not a frame of f is also a window depends solely on the restriction of f to the support of the frame. To decide about the future of a window of f, let $[x, y]$ be its support. This window is also a window of g if $y < z$, and it is also a window of h if $z < x$. The windows that need attention are the ones with $x < z < y$. A triple-panel window consists of three panels, so we distinguish between three cases:

- $W(a, b)$ suffers an *injury* if z cuts through the in-panel. Then a and b lie on the same side of z and they still span a window, albeit with short wave.
- $W(a, b)$ suffers a *fatality* if z cuts through the mid-panel. Then a and b lie on different sides of z and need to find new partnering critical items to span new windows.
- $W(a, b)$ suffers a *scare* if z cuts through the out-panel. These windows are difficult to find in the banana tree of f, but they are easy to find in the banana tree of $-f$, in which $W(b, a)$ suffers an injury.

The splitting of the banana tree proceeds in three steps: first, find the smallest banana that suffers an injury, fatality, or scare; second, find the remaining such bananas; and third, split the banana tree of f into the banana trees of g and h. We address all three steps and highlight the most interesting feature in each.

The first step is difficult because of the lack of an appropriate search mechanism in the banana tree. To explain this, we recall that the banana tree stores the mirrors implicitly, as the upper ends of the in-trails. Like in the case of an anti-cancellation, the challenging case is when mirrors accumulate and we have to locate z in their midst. As before, we locate z by linear search, scanning the mirrors in the order of decreasing function value. In contrast to the anti-cancellation, we can now charge the cost for the search to the changes in the extended persistence diagram. Indeed, every mirror we pass belongs to a window that experiences a scare. Such a window of f is neither a window of g nor of h, so its spanning critical items will re-pair and span new windows after the operation.

The second step traverses a path upward from the smallest affected banana we just identified. In each step of the traversal, we determine the corresponding window and push it onto the stacks of windows that experience an injury, fatality, or scare. A window that experiences an injury remains a window, but now with short instead of simple wave. Whether this window with short wave belongs to the banana tree of g or that of h depends on whether the spanning minimum is to the right or the left of the spanning maximum. A window that experiences a fatality falls apart, with one of the two spanning critical items in g and the other in h. Finally, a window that experiences a scare stops to be a window, in spite of having both critical points in g or in h. As mentioned earlier, such a window is difficult to find in the banana tree of f, but it is easy to find in the banana tree of $-f$, where it experiences an injury. We thus process both banana trees simultaneously, and distributed the windows or their corresponding bananas as needed. The upward traversal halts when we reach the *spine* of the tree, by which we mean the sequence of left children that start at the root, or the sequence of right children that start at the root. This is important because moving further until we reach the root is not necessary and can be costly because the spine can be arbitrarily long and the steps towards the root cannot be charged to changes in the extended persistence diagram.

The third step re-pairs the critical items of the windows that experience a fatality or scare. All new windows are with short wave, for else they would be windows of f before the splitting, which is a contradiction. Hence, their bananas belong to the spines of the banana trees of g and h. The bananas in the spines have the particularly simple structure that their critical items come in sequence. For the right spine of the banana tree of g, this sequence increases from right to left, for the left spine of the banana tree of h, the sequence increases from left to right, and both are consistent with the sequence in which we collect the corresponding windows in the second step. It follows that the available critical items can be paired up in sequence, which takes $O(1)$ time per item.

Corresponding to the concatenation of two lists, we pairwise *glue* the banana trees of the two maps. The operation is the inverse of splitting, so we omit further details. A final word about the cost paid by the changes in the augmented persistence diagram. How do we compare one diagram with two, which we get after splitting? To deal with this issue, we consider the maps g and h to be *one* map, namely a map on two intervals. Then we still have one augmented persistence diagram and the symmetric difference between the diagrams before and after the operation is well defined.

4 Data Structures

We use five inter-connected data structures to represent a linear list or, equivalently, the implied piecewise linear map: a doubly-linked list of all items, two dictionaries storing the minima and maxima for quick access, and two ordered trees representing the information in the augmented persistence diagram for quick update. The novel aspect is the implementation of the ordered trees as *banana trees*, to be described shortly. There are two such trees, storing the homological critical points of the map and its negation, which are subsets of all items. Indeed, the non-critical items are stored only in the doubly-linked list, but they enter the dictionaries and banana trees when they become critical. The difference between critical and non-critical items is defined in terms of the piecewise linear map implied by the items. Let $m \geq 2$ and write $c_1, c_2, \ldots, c_m$ for the list of values, which we assume are distinct. We construct the map, $f\colon [0, m+1] \to \mathbb{R}$, by setting $f(i) = c_i$, for $1 \leq i \leq m$, and adding artifical ends by setting

$$f(0) = c_1 + \varepsilon \cdot \operatorname{sgn}(c_2 - c_1), \tag{1}$$
$$f(m+1) = c_m + \varepsilon \cdot \operatorname{sgn}(c_{m-1} - c_m), \tag{2}$$

in which $\operatorname{sgn}(c) = \pm 1$ depending on whether $c > 0$ or $c < 0$, and $\varepsilon > 0$ is arbitrarily small and in any case smaller than the absolute difference between any two of the c_i. We call these artificial ends *hooks*; they make sure that items 1 and m are proper minima or maxima.

4.1 Dictionaries

We maintain the items in a *doubly-linked list* ordered according to their position in the interval. For reasons that will become clear shortly, we additionally store the minima and maxima in a *dictionary* each, both ordered like the doubly-linked list. Besides searching, the dictionaries support the retrieval of the minimum or maximum immediately to the left or the right of a given $x \in \mathbb{R}$. It will be convenient to write n for the number of maxima so that $n-1$, n, or $n+1$ is the number of minima. In addition, we will pretend that the items—and sometimes just the minima and maxima—are at consecutive integer locations along the real line. Obviously, this cannot be maintained as we insert and delete items, but it makes sense locally and simplifies the notation and discussions without causing any confusion.

An opportune data structure for the dictionary is a binary search tree, which supports `access`, `insertion`, and `deletion` of an item in $O(\log n)$ time each. Similarly it supports the `cutting` of a dictionary into two, and the `concatenation` of two dictionaries into one—provided all items in one dictionary are smaller than all items in the other—again in $O(\log n)$ time. For comparison, doubly-linked lists can be `cut` and `concatenated` in constant time, provided the nodes where the cutting and concatenation is to happen are given.

4.2 Banana Trees

Let $f\colon [0, m+1] \to \mathbb{R}$ be a generic piecewise linear map constructed from a list of m distinct values. The main data structure consists of two trees, one for f and the other for $-f$, which we explain in three steps. In order to ensure that all critical items are represented in both trees, we require homological critical points of f to also be homological critical points of $-f$, and vice versa. This affects how we treat up- and down-type items: up-type items in f are already homological critical points; in $-f$, however, they are down-type items, which are not homological critical points. We use the hooks at down-type items to treat them as proper maxima, and ignore the hooks at up-type when constructing the trees. To describe the tree for f, let $a_0 < b_1 < a_1 < \ldots < b_n < a_n$ be the homological critical points of f, and note that the a_i are minima—with the possible exception of a_0 and a_n, which may be up-type endpoints that are artificially added as hooks. On the other hand, all b_i are proper maxima.

First step: we construct a full binary tree whose nodes are the homological critical values. Among the many choices, we arrange these values such that

I.1 the in-order sequence of the nodes in the tree is the ordering of the homological critical points in the linear list, i.e. of their position in the interval;

I.2 the nodes along any path from a leaf to the root are ordered by increasing values.

It is not difficult but important to see that there is a unique full binary tree that satisfies Conditions I.1 and I.2. Indeed, the largest value is a maximum, b_j, which is necessarily the root of the tree. The left subtree is the recursively defined tree of $a_0, b_1, \ldots, a_{j-1}$, and the right subtree is the recursively defined tree of $a_j, b_{j+1}, \ldots, a_n$. By induction, the two subtrees are unique, so the entire tree is unique.

Second step: we decompose the tree into edge-disjoint paths, each connecting a leaf to an internal node. Since there is one extra leaf, we add a *special root*, β, whose only child is the root, as an extra internal node. We define β to be greater than all items, both in terms of function value and along the interval; that is: $f(i) < f(\beta)$ and $i < \beta$ for all items i. With this small change, the paths define a bijection between the leaves and the internal nodes. Again there are many choices, and we pick the paths such that

II each path connects a leaf, a, to the nearest ancestor, b, for which a does not have the smallest value in the subtree of b, and to the special root, if no such ancestor exists.

After fixing the tree in the first step, Condition II implies a unique partition of the edges into $n+1$ paths; see Figure 4. Comparing with the windows introduced in Section 2, we note that each path corresponds to a double-panel window: the lowest and highest nodes are the minimum and maximum spanning the window, and all other nodes of the path are maxima of nested windows in the in-panel or the mid-panel of the double-panel window. The out-panel is indirectly represented by the requirement that its highest node is interior to another path whose lowest node has smaller value than the lowest node of the current path. If the window is with simple wave, then the triple-panel window is also represented in the down-tree, except that in- and out-panels are exchanged.

Third step: we split every path into two parallel trails. Let $a = q_0, q_1, \ldots, q_{\ell-1}, q_\ell = b$ be such a path, and recall that $f(q_i) < f(q_{i+1})$ for $0 \leq i \leq \ell - 1$. Assuming $a < b$, the *in-trail* consists of a, every q_i whose right child is q_{i-1}, and b, and the *mid-trail* consists of a, every q_i whose left child is q_{i-1}, and b. The items in the in-trail are smaller than those in the mid-trail, which motivates the alternative terminology of the *left trail* for the former and the *right trail* for the latter. If $a > b$, the in-trail is the right trail and the mid-trail is the left trail, so the in-trail corresponds to the in-panel and the mid-trail to the mid-panel in either case. Both trails *start* at a and *end* at b. All other q_i are *interior* to the trails. For the special root, β, and the corresponding lower end of its path, α, the order is not defined, so we make an arbitrary choice and call the left trail the *in-trail* and the right trail the *mid-trail*. We call a trail *empty* if it contains no interior nodes. In both trails, the items as well as their values are sorted. To state this formally, assume again that $a < b$:

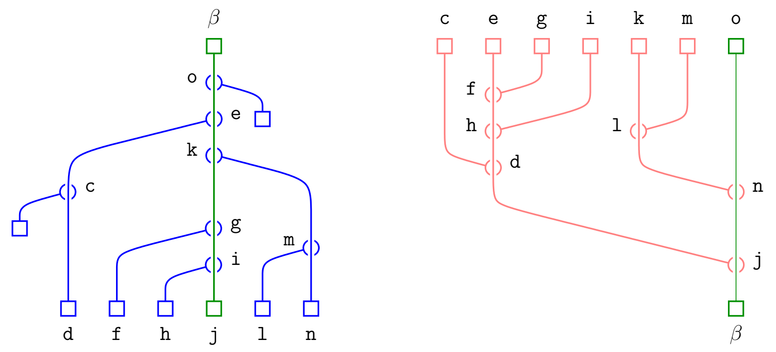

Figure 4: *Left:* the path-decomposition of the binary tree for the map, f, displayed in Figure 2. The nodes in its spine (to be defined shortly) are `c`, `e`, β, and `o`. *Right:* upside-down drawing of the path-decomposition of the binary tree for $-f$. The nodes in its spine are `d`, `j`, and β.

III.1 writing $a = u_0, u_1, \ldots, u_j = b$ for the nodes along the in-trail, we have $u_i > u_{i+1}$ for $0 \leq i \leq j-2$ and $f(u_i) < f(u_{i+1})$, for $0 \leq i \leq j-1$;

III.2 writing $a = v_0, v_1, \ldots, v_k = b$ for the nodes along the mid-trail, we have $v_i < v_{i+1}$ and $f(v_i) < f(v_{i+1})$, for $0 \leq i \leq k-1$.

The respective first inequalities in III.1 and III.2 imply that the nodes of the in-trail precede the nodes of the mid-trail. In contrast, there is no particular order between the values of nodes in different trails. We draw the trails roughly parallel, like the outline of a banana. Letting a, b be the lower and upper ends, we call the pair of trails the *banana* spanned by a, b. Except if b is the special root, b is also internal to another trail, so b is where the banana spanned by a, b connects to another banana; see Figure 5. We summarize the crucial properties of bananas in a lemma whose proof follows directly from the construction and is thus omitted.

Lemma 4.1 (Bananas and Windows)**.** *Let a, b be critical points of a generic map, f, with $f(a) < f(b)$. The two points span a banana in* Up(f) *iff $W(a, b)$ is a window of f. Furthermore, another window, $W(p, q)$, is immediately nested in the in-panel or mid-panel of $W(a, b)$ iff its maximum, q, is an interior node of the in-trail or mid-trail of the banana spanned by a and b, respectively.*

Each node stores pointers to its neighbors along the trails: for a node p, $\mathsf{in}(p)$ and $\mathsf{mid}(p)$ point to the first node on the in- and mid-trails beginning at p, respectively; a maximum q has additional pointers $\mathsf{up}(q)$ and $\mathsf{dn}(q)$, which point to the node above and below q on the same trail. The banana tree uses additional pointers to connect the ends of its trails: letting a and b be the lower and upper ends of a trail, we store $\mathsf{dth}(a) = b$ and $\mathsf{low}(p) = a$, in which p is any node in the banana other than b. For convenience, this includes $p = a$, which thus stores a pointer to itself. To obtain the lower end of a banana from its upper end we define $\mathrm{Bth}(b) = \mathsf{low}(\mathsf{in}(b)) = \mathsf{low}(\mathsf{mid}(b))$. Observe that each of the two banana trees stores exactly one node per critical item and thus requires $O(n)$ space. The dictionaries and the doubly-linked list require additional $O(m)$ space and hence the total space used by our data structure is linear in the total number of items.

We conclude this section with the assertion that the banana tree of a linear list is unique. This will be useful in proving some of the algorithms in the subsequent section correct.

Lemma 4.2 (Uniqueness of Banana Tree)**.** *Given a linear list of distinct values, there is a unique path-decomposed binary tree satisfying Conditions* I.1, I.2, *and* II, *and a resulting unique banana tree satisfying Conditions* III.1 *and* III.2.

Proof. The algorithm that constructs the banana tree of a linear list is deterministic and thus computes a unique such tree, which we denote B. It therefore suffices to prove that no other banana tree satisfies Conditions I.1, I.2. II, III.1, and III.2. We have already seen that there is a unique ordered binary tree that satisfies I.1 and I.2. Similarly, there is a unique decomposition of this tree into paths that satisfies II.

To get a contradiction, assume $B' \neq B$ is another banana tree that satisfies I.1 to III.2. Write B_2 and B_2' for the path-decomposed ordered binary tree we get by merging the two trails of each banana in B and B', respectively. To satisfy Condition I.2, the merging must preserve the ordering by value, and by Conditions III.1 and III.2 this is indeed possible. But the step from B' to B_2' is deterministic, and so is the step from B to B_2. We have already established that $B_2' = B_2$, which implies $B' = B$, as claimed. □

The reverse of Lemma 4.2 does not hold for the trivial reason that the banana trees do not store the non-critical items. There is also the more subtle reason that different path-decomposed ordered binary trees can map to the same banana tree. Indeed, the banana trees do not specify the order of values between parallel trails, so merging them can result in different ordered paths.

4.3 String and Spine

It is possible to connect the trails of a banana tree into a single curve such that the homological critical points are listed from left to right according to their positions in the interval. Considering a banana tree as a graph, the minima and the special root have two neighbors, and the maxima have four neighbors each, so the maxima would appear twice, but one appearance is the mirror of the maximum, namely the upper end of the in-trail. We list the maximum when we reach the upper end of the mid-trail. Equivalently, we adopt the following rule for a banana spanned by a, b: if $b < a$, we first list b, then walk down the interior nodes of the left trail, then list a, and finally walk up the interior nodes of the right trail, and if $b > a$, we do the same except that we list b at the end rather than at the beginning. The sub-bananas are listed recursively when their upper ends are encountered.

As an example consider the banana tree sketched in Figure 5 on the left: after starting at the special root, we first encounter the left hook, then c, d, e and so on until n, o, and finally the right hook before returning to the special root. We call this the *string* of the banana tree. It is not

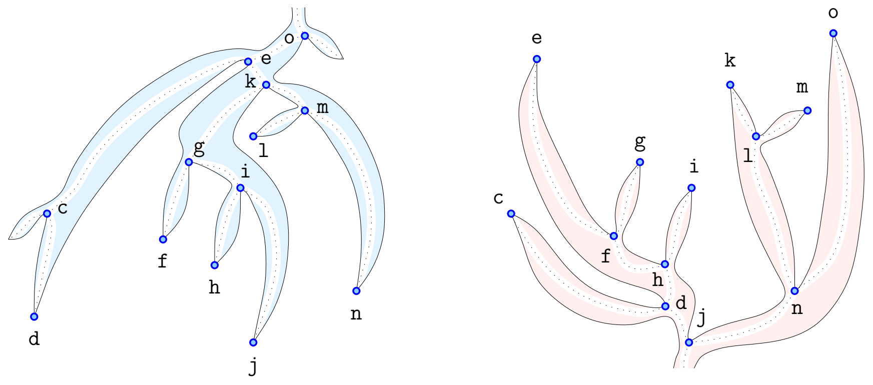

Figure 5: *Left:* the banana tree of the map in Figure 2, with dotted curves showing the tree before splitting its paths; compare with the *left* drawing in Figure 4. *Right:* upside-down drawing of the banana tree for the negated map; compare with the *right* drawing in Figure 4.

difficult to see that this is also the in-order traversal of the tree after the first step of the banana tree construction.

We continue with the definition of an important subset of nodes in a banana tree. The *left spine* consists of the special root, β, as well as the first interior node of every left trail with upper end in the left spine. Symmetrically, the *right spine* consists of β as well as the first interior node of every right trail with upper end in the right spine. The two overlap in β, and the *spine* is the union of the left and the right spines; see Figure 4 for an example. The nodes in the spine can also be characterized in terms of the windows they span.

Lemma 4.3 (Spines and Windows)**.** *Let* $\mathrm{Up}(f)$ *be the banana tree of the map* f*. Then*

(i) the special root β *and the global minimum span the unique global window of* f*;*

(ii) a node $b \neq \beta$ *of the left spine spans a window with wave that is short on the left;*

(iii) a node $b \neq \beta$ *of the right spine spans a window with wave that is short on the right;*

(iv) all other internal nodes of $\mathrm{Up}(f)$ *span windows with simple waves.*

Proof. It is not necessary to give an argument for (i). Since (ii) and (iii) are symmetric, it suffices to prove (ii). By the first step of the construction of a banana tree in Section 4.2, a node b belongs to the left spine iff $f(b) > f(q)$ for all items $q < b$. It follows that the window spanned by b is not constrained on the left, so it extends to the beginning of the list. In other words, the window is with short wave, and the wave is short on the left.

The condition $f(b) > f(q)$ for all $q < b$ is also necessary to have a short wave on the left, so all such windows are spanned by nodes in the left spine. Symmetrically, all windows with short wave on the right are spanned by nodes in the right spine. Hence, all other internal nodes span windows with simple wave, which proves (iv). $\square$

According Lemma 4.3, the special root spans the global window, the remaining nodes in the spine span windows with short wave, and the remaining internal nodes span windows with simple wave. This holds for $\mathrm{Up}(f)$ as well as for $\mathrm{Up}(-f) = \mathrm{Dn}(f)$. Since a window with short wave of f is not a window of $-f$, and vice versa [2, Theorem 4.2], this implies that the min-max pairs with the maximum in the spine (other than the special root) are distinct in the two trees. In contrast, the min-max pairs with the maximum not in the spine are the same in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$.

The algorithms for splitting and gluing banana trees in Section 5.3 need to recognize nodes in the spine. We thus label these nodes and maintain the labeling when changes occur.

5 Algorithms

Call the banana trees of a map and its negative the *up-tree* and *down-tree* of f, denoted $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$. We describe the construction of both trees, the extraction of the augmented persistence diagram from these trees, and operations that maintain the trees subject to local changes of the data. We begin with the construction of the up-tree, which takes time linear in the number of items. Together with the extraction, this gives a linear-time algorithm for the augmented persistence diagram; compare with the algorithm of Glisse [11], which constructs the persistence diagram in linear time but not the augmentation. We prove the correctness of the local and topological maintenance operations in appendices A and B.

5.1 Construction

We explain the construction of the up-tree for a generic piecewise linear map defined by a list of m items: $f(i) = c_i$ for $1 \leq i \leq m$. For convenience, we add items 0 and $m+1$ at the two ends, with $f(0) = \infty$ and $f(m+1) = \infty - 1$. The algorithm processes the map from left to right and uses a stack to store a subset of the minima and maxima so far encountered. Specifically, while processing j, the stack stores pairs $(a_0, b_0), (a_1, b_1), \ldots, (a_k, b_k)$ such that

- $0 = a_0 = b_0 < a_1 < b_1 < a_2 < \ldots < a_k < b_k < j$,
- $f(i) \leq f(b_\ell)$ for all $1 \leq \ell \leq k$ and $b_{\ell-1} < i < j$,
- $f(i) \geq f(a_\ell)$ for all $1 \leq \ell \leq k$ and $b_{\ell-1} < i \leq b_\ell$.

The first two properties imply that the points $(b_\ell, f(b_\ell))$ form a descending staircase in the plane, and all points $(i, f(i))$ with $i < j$ that are not steps of the staircase lie below the staircase. The third property says that item a_ℓ minimizes the value among all items i vertically below the step of b_ℓ.

The only relevant items are the critical points, so we eliminate non-critical items in a preliminary scan and connect the remaining items with prv- and nxt-pointers. Here we consider the artificially added items as maxima, so 0 is the first item in this list, followed by nxt(0), nxt(nxt(0)), etc., until we reach item $m+1$. For later use, each remaining item in the list is provided with the appropriate subset of initially `null` in-, mid-, up-, dn-, low-, and dth-pointers. During the main scan the algorithm maintains unfinished bananas, each corresponding to a pair on the stack, as well as a value A, which, after the current item has been processed, is the item with the minimum value to the right of the top-most item on the stack. Whenever we pass a minimum, j, we set $A = j$, as the immediately preceding item must have been a maximum. Whenever we pass a maximum, the stack is accessed through the standard functions Push and Pop, as well as through function Top, which returns the item b at the top of the stack, but without removing the pair (a, b) from the stack.

Suppose (a, b) is the pair at the top of the stack, and we pop it off because $f(b) < f(j)$. If $f(A) < f(a)$, we now know that the unfinished banana spanned by a, b is correct, so we finalize it in FixBanana. Otherwise, A, b span a banana, which must belong to a right trail as $b < A$. We, thus, detach b on the left, attach it on the right, and finalize the banana in FixBanana. Then we set $A = a$ and push (A, j) on the stack after creating its temporary banana and attaching it on the left:

```
$\mathsf{dn}(0) = 1$; PUSH$(a_0 = 0, b_0 = 0)$; $j = 0$;
repeat $j = \mathsf{nxt}(j)$;
        if $j$ is minimum then $A = j$ endif;
        if $j$ is maximum then
           while $f(j) > f(\text{TOP})$ do $(a, b) =$ POP;
                 if $f(A) < f(a)$ then FIXBANANA$(a, b)$
                    else attach $b$ below $j$ on the right;
                         FIXBANANA$(A, b)$; $A = a$
                 endif
           endwhile;
           attach $j$ below $b$ on the left; PUSH$(A, j)$;
           if $j = m + 1$ then FIXBANANA$(A, j)$ endif
        endif
until $j = m + 1$.
```

To "attach j below b on the left", where j is the currently processed item and b is a past item, we create an unfinished banana with upper end j and temporary in- and mid-pointers, and set the relevant pointers of j, as illustrated in Figure 6 on the left:

$\mathsf{up}(j) = b$; $\mathsf{in}(j) = \mathsf{dn}(b)$; $\mathsf{mid}(j) = \mathsf{prv}(j)$; $\mathsf{dn}(j) = \mathsf{nxt}(j)$;
$\mathsf{dn}(b) = \mathsf{up}(\mathsf{in}(j)) = \mathsf{up}(\mathsf{mid}(j)) = j$.

Whenever the pair containing j is later popped off the stack, and a comparison of the values shows that j belongs to a left trail, then these pointers remain unchanged in the final banana.

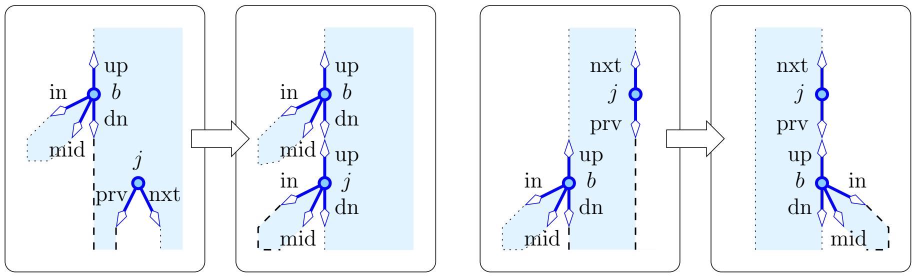

Figure 6: *Left:* item j it temporarily attached on the left, which creates a banana whose upper end, j, is interior to a left trail. *Right:* the temporary attachment of item b on the left is resolved, and b is attached on the right, which creates a banana whose upper end, b, is interior to a right trail.

If, however, this is not the case, we execute "attach b below j on the right", which suitably updates these values to reflect the fact that j belongs to a right trail. Specifically, we detach b on the left and attach it on the right, as illustrated in Figure 6 on the right:

$$\mathsf{dn}(\mathsf{up}(b)) = \mathsf{in}(b); \mathsf{up}(\mathsf{in}(b)) = \mathsf{up}(b);$$
$$\mathsf{up}(b) = j; \mathsf{in}(b) = \mathsf{prv}(j); \mathsf{aux} = \mathsf{dn}(b); \mathsf{dn}(b) = \mathsf{mid}(b); \mathsf{mid}(b) = \mathsf{aux};$$
$$\mathsf{prv}(j) = \mathsf{up}(\mathsf{in}(b)) = b.$$

To fix a banana, we set the low-pointers of its interior nodes and the minimum, as well as the in-, mid-, and dth-pointers of the minimum:

```
function FIXBANANA$(a, b)$:
  $q = b$; $p = \mathsf{in}(b)$; while $p \neq a$ do $\mathsf{low}(p) = a$; $q = p$; $p = \mathsf{dn}(p)$ endwhile; $\mathsf{in}(a) = q$;
  $q = b$; $p = \mathsf{mid}(b)$; while $p \neq a$ do $\mathsf{low}(p) = a$; $q = p$; $p = \mathsf{dn}(p)$ endwhile; $\mathsf{mid}(a) = q$;
  $\mathsf{low}(a) = a$; $\mathsf{dth}(a) = b$.
```

To argue the correctness of the algorithm, we formulate three invariants maintained by the algorithm. Let j denote the current item, and distinguish between *past items*, $b < j$, and *future items*, $b > j$.

(i) The prv-pointers of the future items are unchanged, except possibly the first one, which points to j. In contrast, all nxt-pointers remain unchanged throughout.

(ii) If a past item, b, is on the stack, then the banana rooted at b is temporary and unfinished. The latter means that the in- and mid-pointers of b and the up- and dn-pointers of b and all interior nodes are in place, but not necessarily the remaining pointers that define the banana. In contrast, all its sub-bananas are complete.

(iii) Right before taking b off the stack, it satisfies (ii) except that the banana rooted at b is no longer temporary. After taking b off the stack, its banana is complete, which means that all pointers of its nodes are in place.

It is easy to see that (i) is maintained, as there is only one place where a prv-pointer is altered, and there is no place where a nxt-pointer is altered. Invariant (ii) holds because all interior nodes of the banana rooted at b have been taken off the stack in the past, and their bananas are complete by Invariant (iii). In particular, this implies that each unfinished banana spanned by a, b has a path from $\mathsf{in}(b)$ along dn-pointers to a defining its in-trail, and similarly from $\mathsf{mid}(b)$ along dn-pointers to a defining its mid-trail. Assuming (ii), function FixBanana adds the necessary pointers to complete the banana, which implies (iii).

Extraction. We next discuss how to compute the augmented persistence diagram from the up- and the down-tree of f. Each banana in $\mathrm{Up}(f)$ corresponds to a point in $\mathsf{Ord}(f)$, with the exception of the banana of the special root, which corresponds to a point in $\mathsf{Ess}(f)$. We find these points and the nesting relation by recursively walking along the trails from bottom to top. We are handed the upper end of each pair of trails, so first we jump to the lower end. The recursive function that enumerates all points and arrows is called with two parameters: the upper end of two parallel trails, and the point in the persistence diagram that corresponds to the pair of trails that contain that upper end as an interior node. Initially, the upper end is the special root, and the point is empty:

```
function Walk($b, pnt$):
  $a = \mathrm{Bth}(b)$; output $\mathrm{Point}(a,b) = (f(a), f(b))$ and $\mathrm{Arrow}(a,b) = (\mathrm{Point}(a,b), pnt)$;
  $x = \mathsf{in}(a)$; while $x \neq b$ do Walk($x, \mathrm{Point}(a,b)$); $x = \mathsf{up}(x)$ endwhile;
  $x = \mathsf{mid}(a)$; while $x \neq b$ do Walk($x, \mathrm{Point}(a,b)$); $x = \mathsf{up}(x)$ endwhile.
```

To complete the augmented persistence diagram, we also construct $\mathrm{Dn}(f)$ and apply the recursive function to its parallel trails, with the only difference that the banana of the special root does not correspond to any point in $\mathsf{Dgm}(f)$.

Summary. After removing all non-critical items, which takes $O(m)$ time, the construction of the two banana trees as well as the extraction of the augmented persistence diagram takes only $O(n)$ time. Indeed, the main scan of the construction algorithm completes each banana only once, and altogether touches each item only a constant number of times. We summarize our findings for later reference.

Theorem 5.1 (Time to Construct). *Let $f\colon [0, m+1] \to \mathbb{R}$ be a generic piecewise linear map with n maxima. After removing all non-critical items in $O(m)$ time, $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$ can be constructed in $O(n)$ time, and the augmented persistence diagram of f can be computed from these trees in $O(n)$ time.*

5.2 Local Maintenance

Recall that the banana trees store all critical items but none of the non-critical items. As a temporary exception, we *delete* items by first adjusting their values until they become non-critical. To formalize the operation that *adjusts* the value of an item, j, from c_j to d_j, we write $f, g\colon [0, m+1] \to \mathbb{R}$ for the maps before and after adjustment, so $g(i) = f(i)$ for $0 \leq i \neq j \leq m+1$, and $f(j) = c_j$, $g(j) = d_j$. To avoid complications near the endpoints, assume $3 \leq j \leq m-2$. We give details on how to treat endpoints in appendix A.6. We modify the trees while following the *straight-line homotopy* from f to g, which is the 1-parameter family of maps $h_\lambda\colon [0, m+1] \to \mathbb{R}$ defined by $h_\lambda(x) = (1-\lambda)f(x) + \lambda g(x)$ for $0 \leq \lambda \leq 1$. Clearly, $h_0 = f$ and $h_1 = g$.

The homotopy reduces to a sequence of *interchanges*—of two maxima or two minima—followed or preceded by a *cancellation* or an *anti-cancellation*, which reflect the disappearance or appearance of a point into or from the diagonal of the persistence diagram, or a *slide*, which occurs when a critical item becomes non-critical due to a non-critical neighbor becoming critical. We will discuss these operations in detail below. Among the adjustments, we consider two scenarios:

A: item j is non-critical in f and increases its value;

B: item j decreases its value until it becomes non-critical in g.

The scenario in which j is non-critical and decreases its value is symmetric to A, and either of the two is applied after we insert a new item. The scenario in which j increases its value until it becomes non-critical is symmetric to B, and either of the two is needed before we delete an item. Other adjustments are subsequences or compositions of Scenarios A and B or their symmetric versions.

Scenario A. We increase the value of a non-critical item, j, and we assume that it becomes critical, else up-tree and down-tree would not be affected. We consider the case in which $f(j-1) < f(j) < f(j+1)$ and $f(j-1) < g(j) > f(j+1)$, i.e., the item j becomes a maximum in g. If $j+1$ is non-critical in f, it becomes a minimum in g and the number of critical items increases by two. If $j+1$ is critical (a maximum), then it becomes non-critical in g and item j replaces it as maximum; the number of critical items does not change. In the former case we perform an anti-cancellation to introduce the banana spanned by $j+1$ and j; in the latter case we perform a slide. Afterwards we fix the position of j in the up-tree and down-tree by performing a sequence of interchanges:

```
if $g(j) > f(j+1)$ then
  if $f(j+1) < f(j+2)$ then anti-cancel $j$ and $j+1$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$
                        else slide $j+1$ to $j$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$
  endif;
  set $q = \mathtt{up}(j)$ in $\mathrm{Up}(f)$;
  while $f(q) < g(j)$ do interchange $j$ and $q$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$;
                       $q = \mathtt{up}(j)$ in $\mathrm{Up}(f)$
  endwhile
endif.
```

Note that we iterate with $q = \mathtt{up}(j)$ and not with $q = \mathtt{up}(q)$. This is not a mistake because the interchange in $\mathrm{Up}(f)$ is of two maxima, j and q, which swaps the two nodes (see below). In contrast, the interchange in $\mathrm{Dn}(f)$ is of two minima, albeit they are the same two items, j and q. Performing these interchanges simultaneously, we save the effort of independently finding the next relevant interchange of minima, which would be costly. The correctness of this strategy is guaranteed by Lemma 5.2, which we will state and prove after formalizing the notion of an interchange.

Scenario B. The symmetric version of Scenario A—in which j decreases its value—is implemented accordingly, by switching the roles of the up-tree and the down-tree. We use the simultaneous interchange of maxima and minima also in the implementation of this inverse of Scenario A. There are again symmetric cases, and we consider the case in which $f(j-1) < f(j+1)$ and $f(j-1) < f(j) > f(j+1)$. Furthermore, we assume that item j is interior to a left trail. The operation begins with a sequence of interchanges of maxima in the up-tree and of minima in the down-tree, followed by a cancellation or by a slide:

```
loop $q = \arg\max\{f(\mathsf{dn}(j)), f(\mathsf{in}(j)), f(\mathsf{mid}(j))\}$ in Up($f$);
  if $f(q) > f(j+1)$ then interchange $q$ and $j$ in Up($f$) and Dn($f$)
                   else exit endif
forever;
if $f(j+1) < f(j+2)$ then cancel $j$ with $j+1$ in Up($f$) and Dn($f$)
                   else slide $j$ to $j+1$ in Up($f$) and Dn($f$)
endif.
```

Note that q is either a maximum or its value is less than $f(j+1)$ as $f(j-1) < f(j+1)$, so we will not attempt to interchange a maximum, j, with a minimum, q, which would be impossible indeed. We continue with the details for the interchanges and (anti-)cancellations.

Interchange of maxima. An interchange between two maxima with $f(j) < f(q)$ has structural consequences only if $q = \mathsf{up}(j)$, as this is the only case in which a uniqueness condition, namely III.1 or III.2, is violated. We will see that the algorithm does not ever get into the situation of attempting any other interchange. Assume $q < j$, which is the situation depicted in the top row of Figure 7. Let i and p be such that $j = \mathsf{dth}(i)$ and $q = \mathsf{dth}(p)$. If $j = \mathsf{dn}(q)$, we distinguish two cases depending on the order of $f(i)$ and $f(p)$.

Case 1: $f(i) < f(p)$. Remove q from its trail and add it right below j in j's in-trail, as illustrated in Figure 7 in the top left. Adjust the pointers of the involved nodes accordingly.

Case 2: $f(i) > f(p)$. Exchange j and q as upper ends of their respective bananas, remove q from its trail and add it right below j in j's mid-trail, as illustrated in Figure 7 in the top right. Adjust pointers, and in particular set $\mathsf{dth}(i) = q$ and $\mathsf{dth}(p) = j$. The in-trail of i becomes its mid-trail and vice versa.

In both cases, j joins the left spine of the up-tree iff q is a node of the left spine already before the operation. The cases where $j < q$ and either $j = \mathsf{mid}(q)$ or $j = \mathsf{in}(q)$ are similar to the reverse of the cases with $q < j$ and $j = \mathsf{dn}(q)$, and are illustrated in the bottom row of fig. 7. There are also the inverse operations (reading Figure 7 from right to left), which apply when $j = \mathsf{up}(q)$, i.e., the case $f(j) > f(q)$, and the symmetric cases.

Interchange of minima. An interchange of maxima in Up(f) is always done in parallel with an interchange of minima in Dn(f). It can, however, happen that the interchange of j and q has a structural effect on Up(f) but not on Dn(f). An example is the interchange of nodes i and g in Figure 4 and 5. They are internal nodes in the up-tree, so the effect of the interchange is as depicted in Figure 7 on the left. The two nodes are leaves in the down-tree whose bananas do not meet, so the interchange of minima has no effect.

In general, the interchange of two minima with $f(j) < f(q)$ has structural consequences only if $p = \mathsf{dth}(q)$ is interior to the banana spanned by j and $i = \mathsf{dth}(j)$, as this is the only case in which $g(j) > g(q)$ leads to a violation of the uniqueness conditions, namely of II. To describe how the

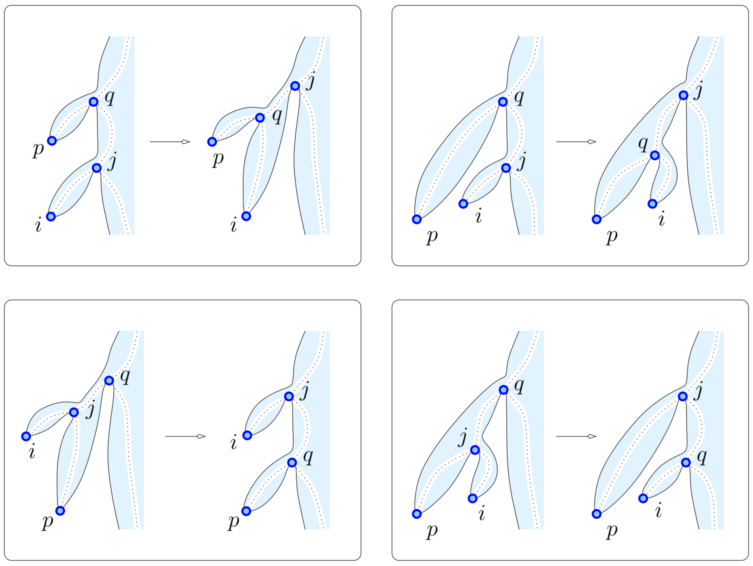

Figure 7: The interchange of two maxima, j and q. In the *dotted* underlying tree, the operation corresponds to a rotation. Before the interchange, the pairs are i, j and p, q. *Top left:* $f(i) < f(p)$ which preserves the pairs. *Top right:* $f(i) > f(p)$ and the pairs change to i, q and p, j. *Bottom left:* $j = \mathsf{in}(q)$ which preserves the pairs. *Bottom right:* $j = \mathsf{mid}(q)$ and the pairs change to i, q and p, j.

trails are updated, we assume that p is interior to the left trail. We split this trail into the upper part above p and the lower part below p (with neither part including p). Similarly, we split the parallel right trail connecting i to j into upper and lower, with values larger and smaller than $f(p)$, respectively. Then we concatenate the upper part of the left trail with the left trail connecting p to q (but without the upper end, which is p), and we concatenate the upper part of the right trail with the right trail connecting p to q (this time including p). Finally, we join the two lower parts to form parallel trails connecting p to j.

Observe that splitting the left trail is easy because it contains p as an interior node. We split the right trail by traversing it one node at a time from the upper end until we reach the first node with value less than $f(p)$. Note that all the travsersed node on the right trail will have a change in their low-pointer and we will use this fact when we analyze the running-time of the operation at the end of this subsection.

Simultaneity of interchanges. Next, we prove the correctness of coupling interchanges as described in Scenarios A and B. The moment two maxima of f interchange is of course also the moment at which they interchange as minima of $-f$. However, many interchanges are irrelevant, in the sense that they cause no structural changes to the banana trees. Many of these irrelevant interchanges go unnoticed by our algorithm (and fortunately so), but it is important that no relevant interchange is overlooked. We claim that every relevant interchange of minima corresponds to a relevant interchange of maxima.

To formalize this claim, let $F\colon [0, m+1] \to \mathbb{R}$ be a piecewise linear map determined by its values at the integers, and assume that these values are distinct, with the exception of $F(j) = F(q)$ at maxima $j \neq q$ of F. Let $f, g\colon [0, m+1] \to \mathbb{R}$ be piecewise linear maps defined by the same values at the integers, except that $f(j) = F(j) - \varepsilon$ and $g(j) = F(j) + \varepsilon$ for a sufficiently small $\varepsilon > 0$. The straight-line homotopy from f to g is an interchange of maxima, namely of j and q, and that from $-f$ to $-g$ is an interchange of minima, again of j and q. We call the former *relevant* if $\mathrm{Up}(f)$ and $\mathrm{Up}(g)$ differ by a rotation, namely of j and q, and we call the latter *relevant* if $\mathrm{Dn}(f)$ and $\mathrm{Dn}(g)$

differ by a change in the pairing.

Lemma 5.2 (Coupling of Interchanges)**.** *Let $f, g\colon [0, m+1] \to \mathbb{R}$ as introduced above. If the straight-line homotopy from $-f$ to $-g$, which is an interchange of minima, is relevant, then so is the straight-line homotopy from f to g, which is an interchange of maxima.*

Proof. We prove the contrapositive: that irrelevant interchanges of maxima imply irrelevant interchanges of minima. The interchange of the maxima $j \neq q$ of f is irrelevant if there is a banana in $\mathrm{Up}(f)$ so that j and q belong to opposite trails, or to sub-bananas rooted on opposite trails of this banana. Let this banana be spanned by a, b. Assuming $j < q$, this implies $j < a < q$ and b is either to the left or the right of the three items. Letting i and p be the lower ends of the bananas spanned by j and q, respectively, we observe that $i < a < p$, $f(a) < f(i)$, and $f(a) < f(p)$.

For the negated function, j, q are minima and i, a, p are maxima satisfying $i, j < a < p, q$, $-f(a) > -f(i)$, and $-f(a) > -f(p)$. If $W(i,j)$ and $W(p,q)$ are both with simple waves, then $W(j,i)$ and $W(q,p)$ are triple-panel windows of $-f$ that are separated by a. It follows that the interchange of minima is irrelevant. Similarly, if one or both of these windows are with short wave, then the separation by a implies that the pairing stays constant, so again the interchange is irrelevant. □

There is a special case to consider when the separating banana is spanned by the special root, and j, q are the maxima with the two largest values. For f, we have $\beta = q$ and for g we have $\beta = j$. In words, the interchange of the maxima j and q does not change the pairing but it causes a replacement of the special root.

Cancellations. The last step in Scenario B is the cancellation of items j and $j+1$, which is implemented by removing the two nodes from the up-tree and down-tree. We argue that the implementation is really this easy.

Recall that the value of j during the homotopy from f to g is $h_\lambda(j) = (1-\lambda)f(j) + \lambda g(j)$. By assumption, $f(j-1) < f(j+1)$, so the cancellation happens at $f(j-1) < g(j) < f(j+1)$. Since $f(j) > f(j+1) > g(j)$ and $f(j+2) > f(j+1)$, there exists $0 \leq \mu \leq 1$ such that $f(j+2) > h_\mu(j) > f(j+1)$. At this stage of the homotopy, $j+1$ is a child of j and $\mathtt{dth}(j+1) = j$ in $\mathrm{Up}(f)$. In other words, $j+1$ and j are the upper and lower ends of a pair of empty trails. We can therefore simply remove this pair of trails, while forming a direct link between $\mathtt{dn}(j)$ and $\mathtt{up}(j)$. The situation is symmetric in $\mathrm{Dn}(f)$.

Anti-cancellations. The first step in Scenario A is the anti-cancellation of items j and $j+1$. We describe how to perform the anti-cancellation in the up-tree; the anti-cancellation in the down-tree is symmetric. By assumption, $f(j-1) < g(j) > f(j+1) < f(j+2)$ and $g(j)$ is such that there is no item with value between $f(j+1)$ and $g(j)$. We first identify the maximum b closest to j such that the new minimum $j+1$ lies between j and b, i.e., $j < j+1 < b$. Note that there can be no other critical point between $j+1$ and b.

To insert j, we walk along the path from b to the closest minimum a with $a < j < j+1 < b$, until we find a node q with smaller value than j. The node j is then inserted as a parent of q:

```
if $\mathrm{Bth}(b) < j < b$ then $q = \mathtt{mid}(b)$ else $q = \mathtt{dn}(b)$ endif;
while $f(q) > g(j)$ do $q = \mathtt{in}(q)$ endwhile;
if $q$ is a leaf then if $q = \mathrm{Bth}(b)$ then insert $j$ as $\mathtt{mid}(q)$
                                              else insert $j$ as $\mathtt{in}(q)$
                      endif
                 else insert $j$ as $\mathtt{up}(q)$
endif.
```

After inserting j, we construct a banana spanned by $j+1$ and j, which includes setting $\mathsf{in}(j+1) = \mathsf{mid}(j+1) = \mathsf{dth}(j+1) = j$ and $\mathsf{in}(j) = \mathsf{mid}(j) = j+1$.

Slides. Consider the case in which the value of a maximum, j, is decreased, and assume that $f(j-1) < f(j+1) > f(j+2)$. Note that $j+2$ is non-critical. If $f(j-1) < g(j) < f(j+1)$, then $j+1$ becomes a maximum and j becomes non-critical. The number of critical items remains the same in f and g, but criticality is transferred from one item in f to a neighboring item in g. As mentioned earlier, this is what we call a *slide*.

The same situation occurs when (1) the value of a minimum increases above the value of a non-critical neighbor, (2) the value of a non-critical item increases above the value of a neighboring maximum, or (3) the value of a non-critical item decreases below the value of a neighboring minimum. A slide has no impact on the structure of the banana-tree and only requires to update the association between items and nodes.

Summary. We summarize the findings in this subsection by stating the running-time for adjusting the value of an item in terms of the number of critical points and the number of changes caused to the augmented persistence diagram.

Theorem 5.3 (Time to Adjust). *Let $f\colon [0, m+1] \to \mathbb{R}$ be a generic piecewise linear map with n maxima. The time to adjust the value of an item is $O(\log n + k')$ or $O(\log n + k)$, depending on whether or not the adjustment requires an anti-cancellation, in which k' and k are the differences in the transitive closure of the nesting relation and the augmented persistence diagrams, before and after the adjustment, respectively.*

Proof. We prove the claimed bound on the running-time by charging most steps to the change in augmented persistence diagrams they cause.

An *interchange of maxima* reduces to a rotation plus possibly a swap in the pairing. The rotation takes $O(1)$ time, which we charge to the changing arrow, and the swap takes $O(1)$ time, which we charge to the two points in the diagram that exchange coordinates. An *interchange of minima* reduces to a swap in the pairing followed by resetting the low-pointers along the path connecting the two maxima involved in the swap. As before, the swap is charged to the two points that exchange coordinates, and each resetting of a low-pointer is charged to the corresponding change in the arrow. It is also possible that the interchange of minima has no effect on the binary tree, namely when the corresponding paths are disjoint. This is detected in $O(1)$ time, and this time is charged to the changes caused by the simultaneous interchange of maxima in the other banana tree. A *cancellation* takes $O(1)$ time, and there are at most two cancellations per value adjustment of an item. In contrast, an *anti-cancellation* takes $O(\log n)$ time to find the maximum, b, closest to the new pair of critical items. The while-loop in the anti-cancellation takes as many iterations as there are nodes on the path from b to the newly inserted node. For each of those nodes an arrow appears in the transitive closure of the nesting relation. Thus, the time to insert the new banana is $O(k')$. The insertion itself takes constant time.

In addition, we take $O(\log n)$ time to update the dictionaries whenever an item changes from critical to non-critical, or the other way round, and there are only $O(1)$ such changes per value adjustment of an item. All this adds up to $O(\log n + k')$ time, if the adjustment requires an anti-cancellation, or $O(\log n + k)$ time, if it does not. □

5.3 Topological Maintenance

Given a collection of linear lists, we next study the maintenance of the augmented persistence diagram subject to cutting and concatenating the lists. Recall that a list $c_1, c_2, \ldots, c_m$ induces a piecewise linear map, $f\colon [0, m+1] \to \mathbb{R}$, with $f(i) = c_i$ for $1 \leq i \leq m$. The values at $i = 0, m+1$ are added to create the computationally convenient hooks introduced in Section 4. To *cut* f, we split the list into $c_1, c_2, \ldots, c_\ell$ and $c_{\ell+1}, c_{\ell+2}, \ldots, c_m$ and let $g\colon [0, \ell+1] \to \mathbb{R}$ and $h\colon [\ell, m+1] \to \mathbb{R}$ be the corresponding piecewise linear maps. We need at least two items to construct the hooks, so we require $2 \leq \ell \leq m-1$. We describe the operation for the up-tree, and consider the down-tree only to the extent it provides information to update the up-tree. For ease of reference, we write x for the midpoint between ℓ and $\ell+1$ and set $f(x) = \frac{1}{2}(f(\ell) + f(\ell+1))$.

Splitting a banana tree. We introduce terminology before describing the splitting algorithm in three steps. A banana suffers an *injury*, *fatality*, *scare* if x cuts through the in-, mid-, out-panel of the window, respectively; see Figure 8. A triple-panel window with simple wave is shared by f and $-f$, except that in- and out-panels are exchanged. Hence, a scare in $\mathrm{Up}(f)$ is an injury in $\mathrm{Dn}(f)$, so we exploit the down-tree to find the scares in the up-tree.

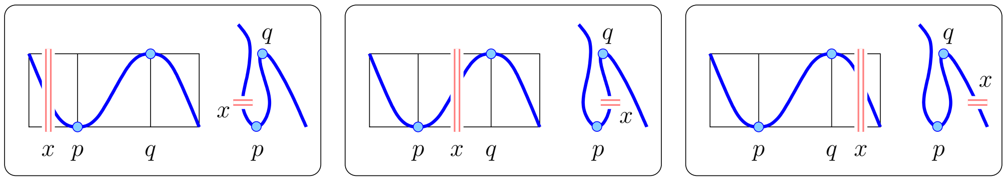

Figure 8: *From left to right:* an injury, fatality, and scare. Correspondingly, x cuts the banana in its in-trail, in its mid-trail, or after the maximum but still above the value of the minimum.

Step 1: **Find smallest banana.** To enumerate the bananas that suffer injuries or fatalities, we first find the smallest such banana. To this end, we search the dictionaries to locate the smallest interval between two critical points that satisfy $a < x < b$. Note however that neither b is necessarily the upper end nor a is necessarily the lower end of this banana. Assuming b is a maximum, we use it to find the smallest banana in the up-tree such that x lies between its lower and upper ends. The search uses a pair of nodes, q, r, maintaining that the upper end of the desired smallest banana has an ancestor that is an interior node of the right trail with upper end q, and r with $f(r) > f(x)$ is a node on this right trail. The iteration advances q and r toward x and halts when these conditions fail. To repeatedly go down the right trail of r, which itself is interior to a right trail, we use the $\mathsf{in}(r)$ pointer. The iteration is slightly different in the first case, when b is interior to a right trail, and in the second case, when b is interior to a left trail:

```
function SmallestBanana($x$):
  find $a < x < b$ and assume that $b$ is a maximum;
  if $\mathsf{dn}(b) < b$ then $q = \mathsf{dth}(\mathsf{low}(b))$; $r = \mathsf{dn}(b)$ else $q = b$; $r = \mathsf{mid}(b)$ endif;
  while $r \neq a$ and $f(x) < f(r)$ do $q = r$; $r = \mathsf{in}(r)$ endwhile;
  return $(\mathrm{Bth}(q), q)$.
```

The time is $O(\log n + k)$, in which k is the number of inspected bananas. Each of these bananas causes a change in the augmented persistence diagram, which pays for the visit.

Step 2: **Stack bananas.** After locating the smallest banana in $\mathrm{Up}(f)$ that suffers an injury or fatality, we find the others by traversing the tree upward. In the process, we load three initially empty stacks with the injuries and fatalities. In doing so, we distinguish spanning min-max pair to the left of x, separated by x, to the right of x, and denote the corresponding stacks L_{up}, M_{up}, R_{up}, respectively. On its way up, the algorithm pushes each banana on one of the three stacks, until it encounters the first banana with maximum in the spine.

```
function LoadStacks ($x$):
  $(p, q) = \text{SmallestBanana}(x)$;
  loop case $p < x$ and $q < x$: $\text{Push}(L_{\text{up}}, (p, q))$;
            $p < x$ xor $q < x$: $\text{Push}(M_{\text{up}}, (p, q))$;
            $p > x$ and $q > x$: $\text{Push}(R_{\text{up}}, (p, q))$
       endcase;
       if $q$ is in spine of $\text{Up}(f)$ then exit endif;
       $p = \text{low}(q)$; $q = \text{dth}(p)$
  forever.
```

Similarly, we load the initially empty stacks L_{dn}, M_{dn}, R_{dn} with bananas in $\text{Dn}(f)$. When we split the up-tree, it will be convenient to reverse the pairs defining the bananas in the down-tree, and when we split the down-tree, we reverse the pairs defining the bananas in the up-tree. We call a stack with bananas (p_0, q_0) at the bottom to (p_j, q_j) at the top *sorted* if $f(p_j) < \ldots < f(p_0) < f(q_0) < \ldots < f(q_j)$. The stacks satisfy the following properties:

Lemma 5.4 (Sorted Stacks). *Let $c_1, c_2, \ldots, c_m$ be a sequence of distinct values, f the thus defined piecewise linear map, x such that $c_2 < x < c_{m-1}$, and $L_{\text{up}}, M_{\text{up}}, R_{\text{up}}, L_{\text{dn}}, M_{\text{dn}}, R_{\text{dn}}$ the stacks as returned by* LoadStacks *holding the bananas with injury, fatality, and scare in* $\text{Up}(f)$ *and* $\text{Dn}(f)$. *Then*

(i) all six stacks are sorted;

(ii) all bananas on the stacks are with simple wave, except for the last banana pushed onto $L_{\text{up}}, M_{\text{up}}, R_{\text{up}}$, whose maximum belongs to the spine of $\text{Up}(f)$*, and the last banana pushed onto $L_{\text{dn}}, M_{\text{dn}}, R_{\text{dn}}$, whose maximum belongs to the spine of* $\text{Dn}(f)$*;*

(iii) the set of simple wave bananas on M_{up} is the same as that on M_{dn};

(iv) L_{up}, M_{up}, L_{dn} can be merged into a single sorted stack, and so can R_{up}, M_{up}, R_{dn} and L_{up}, M_{up}, R_{up} and $L_{\text{dn}}, M_{\text{dn}}, R_{\text{dn}}$.

Proof. Property (i) is implied by Condition II on the path-decomposition and the fact that the algorithm visits the bananas from bottom to top. Property (ii) holds because all bananas are with simple wave, other than the ones with maxima in the spine, and except for the respective last ones, no banana pushed onto the stacks have their maxima in the spine. To see Property (iii) recall that a simple wave of f is also a simple wave of $-f$. Hence, simple wave bananas belong to both trees, except that they share the mid-trail but not the in-trail. Each banana on M_{up} and M_{dn} suffers a fatality, which cuts cuts the mid-trail and therefore also the corresponding banana on the other stack. Property (iv) follows from the fact that all bananas on one of these stacks must correspond to nested windows. Note, however, that it is not true that all six stacks can be merged to a sorted stack. The reason is that triple-panel windows of bananas in L_{up} and R_{dn} can overlap without being nested, and so can triple-panel windows of bananas in L_{dn} and R_{up}; see the windows spanned by `f`, `g` and `h`, `i` in Figure 2. However, if x cuts through any two such overlapping and not nested windows, then it also separates the critical points that span one from those that span the other. Such pairs neither exist for $L_{\text{up}}, M_{\text{up}}, L_{\text{dn}}$, nor for any of the other three triplets of stacks. □

After pushing the bananas onto the stacks, we remove them from the top. Write $(p, q) = \text{Top}(L_{\text{up}})$ for the topmost banana on L_{up}, with $(p, q) = (\texttt{nil}, \texttt{nil})$ if L_{up} is empty, and similarly for the other stacks. We set $f(\texttt{nil}) = -\infty$. The overall top banana is the one whose maximum has the largest value and is returned by function TopBanana:

```
function TopBanana:
  $Stack$ = nil; $(p,q)$ = (nil, nil);
  for each $S \in \{L_{\rm up}, R_{\rm up}, M_{\rm up}, L_{\rm dn}, R_{\rm dn}\}$ do $(p',q') = $ Top($S$);
      if $f(q') > f(q)$ then $Stack = S$; $(p,q) = (p',q')$ endif
  endfor; return ($Stack$, $(p,q)$).
```

Step 3: **Split up-tree.** This operation splits the up-tree into two and in the process adds at most four new nodes: a second special root, a minimum and a maximum if ℓ and $\ell+1$ are non-critical points prior to the cut, and the up-type endpoint of the two new hooks on both sides of the cut. The algorithm uses the loaded stacks to visit the nodes that need repair from top to bottom, i.e. from the largest to the smallest banana as determined by the stacks. The iteration ends when the stacks are empty:

```
function Split($x$):
  LoadStacks($x$);
  loop ($Stack$, $(p,q)$) = TopBanana;
      if $(p,q)$ = (nil, nil) then exit endif;
      case $Stack \in \{L_{\rm up}, R_{\rm up}\}$: DoInjury($p,q$);
           $Stack = M_{\rm up}$: DoFatality($p,q$);
           $Stack \in \{L_{\rm dn}, R_{\rm dn}\}$: DoScare($p,q$)
      endcase; Pop($Stack$)
  forever.
```

Finally, we add ℓ and $\ell+1$ as new nodes if they become critical in the process, and we do the final adjustment to the values of the new hooks. In each iteration, we have two banana trees, one on the left and the other on the right. An injury transfers part of one tree to the other, and so does a fatality. The latter also adjusts the pairing between the critical points, while a scare only adjusts the pairing. To initialize this setting, we construct a second banana tree consisting of a single banana with two empty trails connecting its special root with a dummy leaf, α. For reasons that will become clear shortly, we set $f(\alpha) = f(p_j) - \varepsilon$, in which p_j, q_j are the nodes that span the top banana on the stacks, and $\varepsilon > 0$ is smaller than the difference between the values of any two items. We observe that the top banana in the first iteration either suffers an injury or a fatality. Indeed, if it suffered a scare, then x would cut through its out-panel and therefore lie between q_j and $\mathsf{low}(q_j)$. But then x cuts through the in- or mid-panel of the banana spanned by $\mathsf{low}(q_j)$ and $\mathsf{dth}(\mathsf{low}(q_j))$, and this banana would have been the top banana on the stacks, which is a contradiction. To see the correctness of the splitting operation, we note that Function Split maintains the following invariants:

(i) both banana trees satisfy the uniqueness conditions I.1, I.2, II, III.1, and III.2;

(ii) any node u that is not on any stack and does not have an ancestor on the stacks is in the left tree, if $u < x$, and in the right tree, if $u > x$.

Invariant (ii) implies that once the stacks are empty, all nodes to the left of x are in the left tree and all nodes to the right of x are in the right tree. By invariant (i) the right and left trees are the unique trees representing the maps g and h. Assuming $x < b$ for the root of the banana tree, the first transfer will be from right to left (as in Figure 9), so we let the existing banana tree be the right tree and the banana with the two empty trails the left tree. We are now ready to discuss the actions specific to the injury, fatality, and scare of a banana.

An *injury* occurs when x cuts the in-trail of the banana spanned by p, q. To simplify the discussion, assume $Stack = R_{\text{up}}$, so $x < p < q$, as illustrated in Figure 8, left, and Figure 9, middle and left. The case $Stack = L_{\text{up}}$ and $q < p < x$ is symmetric.

The injury causes a possibly empty portion of the in-trail to split off and append to the rightmost banana spanned by a spine node of the tree on the left; see again Figure 9. The cut off portion consists of all interior nodes $j < x$ and their sub-bananas. Note that it is quite possible that x cuts the string somewhere in a sub-banana with upper end on the in-trail. In this case, the sub-banana does not transfer as its upper end is to the right of x, and it is instead subject to a later repair operation. Because the bananas taken from the stacks are sorted, we have $f(i) > f(\alpha)$ for all transferred nodes i. We maintain α as an up-type endpoint that spans a banana with b, as before the operation.

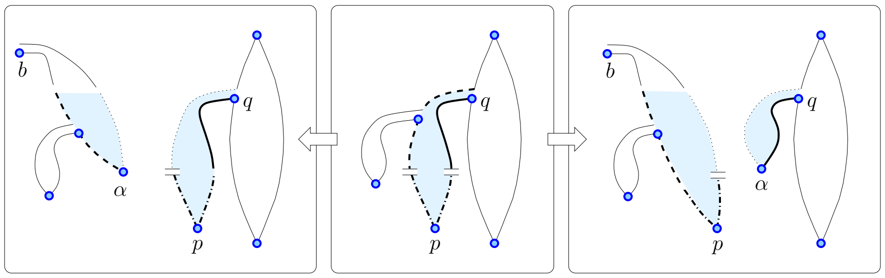

Figure 9: *Middle:* the banana spanned by p, q suffers an injury (cut in in-trail) or a fatality (cut in mid-trail). *Left:* to process the injury, we move the *dashed* portion and append it to the rightmost spine banana of the left tree. The up-type node, α, is paired with the upper end, b, of the expanded banana. *Right:* to process the fatality, we move the entire in-trail together with the *dash-dotted* portion of the mid-trail. The new up-type endpoint, α, is paired with q in the right tree, and p is paired with the upper end, b, of the expanded banana. Dotted portions of trails are empty.

A *fatality* occurs when x cuts the mid-trail of the banana spanned by p, q. Hence, $Stack = M_{\text{up}}$, and we assume $p < x < q$, which is the case illustrated in Figure 8, middle, and Figure 9, middle and right. The case $q < x < p$ is symmetric.

The fatality causes the entire in-trail and a possibly empty portion of the mid-trail to split off and append to the rightmost banana spanned by a spine node of the tree on the left; see again Figure 9. The transfer includes the minimum, p, and all interior nodes, j, with $j < x$. As in the case of an injury, it is possible that x cuts a sub-banana with upper end on the mid-trail. This sub-banana is not transferred and instead subject to later repair operation. Since p moves from the right to the left, we move α from the left to the right and adjust the pairing accordingly: p spans a banana with b in the left tree, and α spans a banana with q in the right tree. To ensure that the path-decomposition remains correct, we set $f(\alpha) = f(p) - \varepsilon$.

A *scare* occurs when x lies in the out-panel of the triple-panel window spanned by p, q, so x cuts neither trail of the corresponding banana. However, there is a banana spanned by q, p in $\text{Dn}(f)$, and x cuts the in-trail of that banana. Assume $Stack = L_{\text{dn}}$, so $p < q < x$, which is the case illustrated in Figure 8, right. The case $Stack = R_{\text{dn}}$ and $x < q < p$ is symmetric.

The scare does not affect the right up-tree, and it affects the left up-tree only indirectly, namely by triggering a change in the pairing. In other words, it preserves the underlying ordered binary tree (First Step of the banana tree construction), but it changes the path-decomposition (Second Step), and therefore also the organization of the bananas (Third Step). This is done by executing an interchange of two minima, namely of p and $\alpha = \mathsf{low}(\mathsf{dth}(p))$. To justify this interchange, we adjust the value of α to $f(\alpha) = f(p) + \varepsilon$.

Gluing two banana trees. Concatenating the lists $c_1, c_2, \ldots, c_\ell$ and $c_{\ell+1}, c_{\ell+2} \ldots, c_m$ is the inverse of cutting $c_1, c_2, \ldots, c_m$ into these two lists. Equivalently, we can think of concatenating $g\colon [0, \ell+1] \to \mathbb{R}$ and $h\colon [\ell, m+1] \to \mathbb{R}$ to get $f = g \cdot h\colon [0, m+1] \to \mathbb{R}$. A triple-panel window with simple wave of g is still a triple-panel window with simple wave of f, and similarly for h and f. Also a triple-panel window whose wave is short at the left end of the domain of g is still a triple-panel window with short wave of f, and similarly for h and the right end of the domain of h. It follows that the only windows that need repair are the global windows of g and h and the windows whose waves are short at the right end of the domain of g or the left end of the domain of h. These windows correspond to the bananas rooted at nodes of the right spine of $\mathrm{Up}(g)$ and the left spine of $\mathrm{Up}(h)$.

In a nutshell, the algorithm visits the nodes in the relevant portions of the two spines from bottom to top. For $\mathrm{Up}(g)$, we get a sequence of nested short wave windows ending with the global window, and we list the corresponding critical points from right to left as $a_0, b_0, a_1, b_1, \ldots, a_i, b_i = \beta_g$. Symmetrically, we list the critical points we get from $\mathrm{Up}(h)$ from left to right as $a'_0, b'_0, a'_1, b'_1, \ldots .a'_j, b'_j = \beta_h$. To have a specific setting, we assume that a_0 is a proper minimum of g (an up-type endpoint after removing the hook), while a'_0 is a hook that is an up-type endpoint of h. Because the windows are nested, we have

$$f(a_i) < \ldots < f(a_0) < f(x) < f(b_0) < \ldots < f(b_i); \tag{3}$$

$$f(a'_j) < \ldots < f(a'_1) < f(x) < f(b'_0) < \ldots < f(b'_j), \tag{4}$$

in which the hook, a'_0, has been removed from the second list; see Figure 10. The two orderings imply that the minimum with largest value is either a_0 or a'_1, and the maximum with smallest value is either b_0 or b'_0. To get a sorted list of nested windows, the algorithm pairs up the lowest maximum with the highest minimum, removes the two, and iterates.

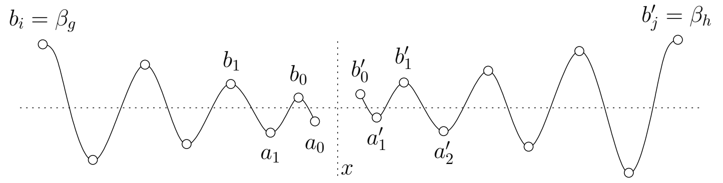

Figure 10: The simplified graph of g to the *left* of x, which connects the critical points that span bananas in the right spine of $\mathrm{Up}(g)$. The symmetrically simplified graph of h to the *right* of x, which connects the critical points that span bananas in the left spine of $\mathrm{Up}(h)$.

Given a_0, the next maximum and then the next minimum to its left are $b_0 = \mathsf{dth}(a_0)$ and $a_1 = \mathsf{low}(b_0)$. Symmetrically, $a'_1 = \mathsf{low}(b'_0)$ and $b'_1 = \mathsf{dth}(a'_1)$. Assuming the configuration in Figure 10, namely $f(a_0) < f(x) < f(b'_0)$, a_0 remains a minimum and b'_0 remains a maximum after connecting the two items monotonically. In other cases, a_0 and b'_0 may become non-critical. As a pre-processing step to gluing, we remove up-type and down-type endpoints that become non-critical when we connect the two lists. Whatever the case, the algorithm assumes that a_0 and b'_0 are the first critical points to the left and right of x, and that a_0 is a minimum and b'_0 is a maximum. We write α for the hook at b'_0 and use it as a dummy leaf, like in cutting. For the purposes of this algorithm we define $f(\beta_h) < f(\beta_g)$ for tie-breaking. We also define $\mathsf{low}(\beta_g) = \mathtt{nil}$ and $\mathsf{low}(\beta_h) = \mathtt{nil}$ with $f(\mathtt{nil}) = -\infty$. In each iteration, the maximum with lower value is paired with a minimum to its left or its right. If the maximum in the left tree, q, is processed, then the candidate minima are $\hat{p} = \mathsf{low}(q)$, which is on the left of q, and p, which is on the same side of q as $\mathrm{Bth}(q)$. If the maximum in the right tree is processed, then the situation is symmetric. The iteration ends when one of the two trees is empty, i.e., consists only of a special root and the dummy α.

```
function GLUE($a_0, b_0', \beta_g, \beta_h$):
  set $q = b_0 = \mathsf{dth}(a_0)$; $q' = b_0'$;
  repeat if $f(q) < f(q')$ then
            $\hat{p} = \mathsf{low}(q)$; if $p \neq \alpha$ then $p = \mathrm{Bth}(q)$ else $p = \mathrm{Bth}(q')$ endif;
            case $f(p) > f(\hat{p})$ and $p = \mathrm{Bth}(q)$: UNDOINJURY$(p, q)$;
                 $f(p) > f(\hat{p})$ and $p = \mathrm{Bth}(q')$: UNDOFATALITY$(p, q)$;
                 $f(\hat{p}) > f(p)$: UNDOSCARE$(\hat{p}, q)$
            endcase; $q = \mathsf{dth}(\mathsf{low}(q))$
          else symmetric cases for $f(q) > f(q')$
        endif
  until $\mathsf{in}(\beta_g) = \mathsf{mid}(\beta_g) = \alpha$ or $\mathsf{in}(\beta_h) = \mathsf{mid}(\beta_h) = \alpha$;
  if $\mathsf{in}(\beta_g) = \mathsf{mid}(\beta_g) = \alpha$ then discard $\beta_g, \alpha$ else discard $\beta_h, \alpha$ endif.
```

The main three subroutines are inverses of the earlier functions and undo injuries, fatalities, and scares. Function UNDOINJURY extends the short wave banana beginning at q to a simple wave banana by inserting maxima into the in-trail beginning at q; see Figure 9 but read the change backward, from the left to the middle. Function UNDOFATALITY extends the short wave banana beginning at q to a simple wave banana while also changing the pairing of q; see again Figure 9 now reading the change from the right to the middle. Finally, function UNDOSCARE adjust the value of the up-type endpoint, $\alpha = \mathrm{Bth}(q)$, to $f(\alpha) = f(\hat{p}) - \varepsilon$, thereby extending the out-panel of the window associated with q and changing the pairing.

Summary. We summarize by stating the running-time for cutting and concatenating lists in terms of the number of critical points and the number of changes to the augmented persistence diagrams.

Theorem 5.5 (Time to Cut and Concatenate)**.** *Let $f\colon [0, m+1] \to \mathbb{R}$ be a generic piecewise linear map with n maxima, and let g and h such that $f = g \cdot h$. The time to cut f into g and h is $O(\log n + k)$, in which k is the size of the symmetric difference between $\overrightarrow{\mathsf{Dgm}}(f)$ and $\overrightarrow{\mathsf{Dgm}}(g) \sqcup \overrightarrow{\mathsf{Dgm}}(h)$, and so is the time to concatenate g and h to form f.*

Proof. We focus on the algorithm for cutting f at x, and omit the argument for glueing, which is symmetric. It takes $O(\log n)$ time to find the two consecutive critical points that sandwich x between then, and it takes $O(\log n)$ time to cut the dictionaries accordingly. The time needed to split $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$ is proportional to a constant plus the number of nodes visited during the iteration.

We argue that we can charge each such node, q, to a change in the augmented persistence diagram in such a way that each change is charged at most twice, once by a node in $\mathrm{Up}(f)$ and once by a node in $\mathrm{Dn}(f)$. Let q be a visited maximum in $\mathrm{Up}(f)$, and let p be the minimum with $\mathsf{dth}(p) = q$. Then $W(p, q)$ is a window with simple or short wave cut by x. If x cuts through the mid-panel or the out-panel, then $W(p, q)$ is neither a window of g nor of h, and we charge q to the disappearance of the point $(f(p), f(p))$ from the ordinary subdiagram. So suppose x cuts through the in-panel. If $W(p, q)$ is with simple wave, $W(q, p)$ is a window of $-f$ and x cuts through its out-panel, so we charge q to the disappearance of $(f(q), f(p))$ from the relative subdiagram. Finally, if $W(p, q)$ is a window with short wave and x cuts through its in-panel, then the algorithm has reached the spine of $\mathrm{Up}(f)$. By Lemma 4.3, all ancestors of q are of the same type, so we can end the traversal here. Thus, this case causes only constant work which is charged to the cutting operation directly. □

We remark that the $O(\log n + k)$ time bound for cutting and concatenating would not hold if in splitting and gluing we traversed the banana trees all the way to the respective special roots.

Let q be the node on the spine where the algorithm halts. The path connecting q to the special root may be arbitrarily long, and none of these nodes corresponds to a change in the augmented persistence diagram. A particular map for which this happens is the *damped sine function*, defined by $f(x) = x \sin x$; see the right half of Figure 10 for a sketch. We get a triple-panel window for each min-max pair, each with short wave on the left. If we cut f close to 0, then all these windows get insubstantially smaller but remain to be spanned by the same min-max pairs. These changes are not visible in the augmented persistence diagrams of f and the functions created by cutting f at x.

6 Discussion

The main contribution of this paper is a dynamic data structure for maintaining the augmented persistence diagram of linear lists. The data structure starts with the Cartesian tree of Vuillemin [18] (see also Aragon and Seidel [1]), which it decomposes into paths and then splits into pairs of parallel trails, arriving at an unconvential representation referred to as banana tree. For the efficiency of the data structure, it is essential to maintain a pair of banana trees, which store complementary information about the list. The most important next step is the implementation of the data structure and its algorithms. With such an implementation, we can deepen our understanding of the persistence of random lists, which in turn can be used as a baseline for our understanding of non-random time series.

The theoretical foundations for our dynamic algorithms are described in [2], where maps on 1-dimensional domains, which include but go beyond intervals are described. Can the banana trees be extended to geometric trees (which allow for bifurcations) and geometric networks (which allow for bifurcations as well as loops) without deterioration of the asymptotic running time? Because of the unconventional representation of trees in terms of bananas (pairs of parallel trails), we finally ask whether there are other dynamic data structure questions that can benefit from this representation.

A Correctness of Local Maintenance

In this section, we prove the correctness of the local operations and of the algorithms for Scenarios A and B (see lemmas A.18 and A.19 in appendix A.6). We also give details on how to treat changes in value that lead to an up-type item becoming down-type or vice versa. We make extensive use of the homotopy $h_\lambda \colon [0, m+1] \to \mathbb{R}$ defined in section 5.2 as $h_\lambda(x) = (1-\lambda)f(x) + \lambda g(x)$ for $0 \leq \lambda \leq 1$, where f and g differ in the value of a single item. The change in function value from f to g can be arbitrary and the transformation of $\mathrm{Up}(f)$ into $\mathrm{Up}(g)$ is achieved by a sequence of local operations, which we define in terms of small changes in the function value of a single item. To make the notion of small change more precise, we define *contiguity* of items in terms of function value.

Definition A.1 (Contiguity)**.** Items p and q are *contiguous in f* if there exists no item $u \neq \{p, q\}$ such that $f(u) \in [\min\{f(p), f(q)\}, \max\{f(p), f(q)\}]$.

Each local operation involving items p and q is then defined as the transformation of $\mathrm{Up}(h_\sigma)$ into $\mathrm{Up}(h_\theta)$, where p and q are contiguous in h_σ and h_θ. We prove the correctness of local operations in terms of such a small change, then show how to extend this to larger changes where items are not necessarily contiguous. This allows us to combine the local operations to form the Scenarios A and B. It is easy to see that when items remain contiguous upon exchanging values then they swap places in the order of items by function value. This is stated in the following lemma.

Lemma A.2 (Local Changes)**.** *Let $0 \leq \sigma < \theta \leq 1$. Let p and q be items that are contiguous in both h_σ and h_θ with $h_\sigma(p) < h_\sigma(q)$ and $h_\theta(p) > h_\theta(q)$. The order of items by function value is the same in h_σ and h_θ except that p and q are swapped.*

In some of the correctness proofs, we analyze how a change in the function value of an item affects the structure of windows and leverage the correspondence between windows and bananas established in lemma 4.1. In other correctness proofs, we use the fact that there is a unique banana tree satisfying the conditions proved in lemma 4.2. To this end we reformulate the uniqueness conditions in terms of individual nodes and their pointers. This requires the following definitions.

Definition A.3 (Ancestor and Descendant)**.** A node p is an *ancestor* of an internal node q if $p = \mathsf{up}^*(q)$ and of a leaf r if $p = \mathsf{up}^*(\mathsf{in}(r))$ or $p = \mathsf{up}^*(\mathsf{mid}(r))$, where up^* denotes a sequence of $\mathsf{up}(\cdot)$-pointers. A node s is a *descendant* of a node t if t is an ancestor of s.

Definition A.4 (Banana subtree)**.** The *banana subtree rooted at a maximum q* is the banana tree consisting of the banana spanned by $\mathrm{Bth}(q)$ and q and all bananas whose maximum has an ancestor other than q on the in-trail or mid-trail starting at $\mathrm{Bth}(q)$ and ending at q.

We now state the new uniqueness conditions as Invariants 1 to 3.

Invariant 1. For each maximum q of f, except for the special root, it holds that

1. if $\mathrm{Bth}(q) < q$, then all nodes $u \neq q$ in the banana subtree rooted at q satisfy $u < q$ and all descendants v of $\mathsf{dn}(q)$, including $\mathsf{dn}(q)$, satisfy $v > q$;
2. if $\mathrm{Bth}(q) > q$, then all nodes $u \neq q$ in the banana subtree rooted at q satisfy $u > q$ and all descendants v of $\mathsf{dn}(q)$, including $\mathsf{dn}(q)$, satisfy $v < q$.

Invariant 2. For each minimum p of f, except for the global minimum, it holds that $f(p) > f(\mathsf{low}(\mathsf{dth}(p)))$.

Invariant 3. For each maximum q of f, except for the special root, it holds that

1. $f(\mathsf{up}(q)) > f(q) > f(\mathsf{dn}(q))$;
2. if $q \neq \mathsf{in}(\mathsf{up}(q))$, then $\mathsf{up}(q) < q < \mathsf{dn}(q)$ or $\mathsf{dn}(q) < q < \mathsf{up}(q)$;
3. if $q = \mathsf{in}(\mathsf{up}(q))$, then either $\mathsf{up}(q) < q$ and $\mathsf{dn}(q) < q$ or $\mathsf{up}(q) > q$ and $\mathsf{dn}(q) > q$.

We show that there is a unique banana tree satisfying Invariants 1 to 3 for a given map f. We first show that a banana tree satisfying Invariants 1 and 3 also satisfies Conditions III.1 and III.2.

Lemma A.5 (Invariants imply Banana Conditions)**.** *A banana tree $\mathcal{B}$ that satisfies Invariants 1 and 3 also satisfies Conditions* III.1 *and* III.2*.*

Proof. Let p and q be a minimum and a maximum spanning a banana in $\mathcal{B}$. Assume $p < q$; the other case is symmetric. The requirement on function values increasing along the trails from p to q follows immediately from $f(\mathsf{dn}(u)) < f(u) < f(\mathsf{up}(u))$. We now show that the trails are also ordered along the interval. By Invariant 1, $\mathsf{mid}(q) < q$ and $\mathsf{in}(q) < q$. Write $p = v_0, v_1, \ldots, v_\ell = q$ for the nodes along the mid-trail with $v_1 = \mathsf{mid}(p)$, $v_{\ell-1} = \mathsf{mid}(q)$ and $v_i = \mathsf{up}(v_{i-1})$ for $2 \leq i \leq \ell$ and $v_i = \mathsf{dn}(v_{i+1})$ for $0 \leq i \leq \ell - 2$. If $v_{\ell-1} = \mathsf{mid}(q)$ is a maximum, then by Invariant 3 and $\mathsf{mid}(q) < q$ it satisfies $\mathsf{dn}(v_{\ell-1}) < v_{\ell-1} < \mathsf{up}(v_{\ell-1}) = q$. If $v_{\ell-2}$ is a maximum, then it in turn satisfies $\mathsf{dn}(v_{\ell-2}) < v_{\ell-2} < \mathsf{up}(v_{\ell-2}) = v_{\ell-1}$ by Invariant 3. Repeating this argument for all v_i for $1 \leq i \leq \ell - 3$ it follows that $p < v_1 < \cdots < v_\ell$.

Write $p = u_0, u_1, \ldots, u_k$ for the nodes along the in-trail with $u_1 = \mathsf{in}(p)$, $u_{k-1} = \mathsf{in}(q)$ and $u_i = \mathsf{up}(u_{i-1})$ for $2 \leq i \leq k$ and $u_i = \mathsf{dn}(u_{i+1})$ for $0 \leq i \leq k - 2$. By Invariant 3 and $\mathsf{in}(q) < q$ we have $u_{k-1} < \mathsf{dn}(u_{k-1})$. Following a similar argument as for the v_i we get $\mathsf{dn}(u_i) > u_i > \mathsf{up}(u_i)$ for all $1 \leq i \leq k-2$ and it follows that $p = u_0 > u_1 > \cdots > u_{k-1}$. Thus, as required by Conditions III.1 and III.2, the trails are ordered along the interval and by function value, i.e., $\mathcal{B}$ satisfies Conditions III.1 and III.2. ☐

The next lemma states that a path-decomposed binary tree can be transformed into a banana tree satisfying the invariants.

Lemma A.6 (Conditions imply Banana Invariants)**.** *If a path-decomposed binary tree T fulfills the Conditions* I.1, I.2 *and* II*, then the banana tree obtained from T as described in section 4.2 satisfies Invariants 1 to 3.*

Proof. Let T be a path-decomposed binary tree satisfying Conditions I.1, I.2 and II and let $\mathcal{B}$ be the banana tree obtained from T as described in section 4.2. We begin by proving that the banana tree $\mathcal{B}$ satisfies Invariant 3. Recall that $\mathcal{B}$ satisfies Conditions III.1 and III.2 and that by lemma 4.2 it is unique. To see that this implies Invariant 3 consider any banana spanned by a minimum a and a maximum b. Assume $a < b$; the argument for $a > b$ is symmetric. Let $a = u_0, u_1, \ldots, u_j = b$ be the nodes along the in-trail and $a = v_0, v_1, \ldots, v_k = b$ be the nodes along the mid-trail. By Conditions III.1, III.2 we have

1. $u_i > u_{i+1}$ for all $0 \leq i \leq j - 2$ and $f(u_i) < f(u_{i+1})$ for all $0 \leq i \leq j - 1$,
2. $v_i < v_{i+1}$ for all $0 \leq i \leq k - 1$ and $f(v_i) < f(v_{i+1})$ for all $0 \leq i \leq k - 1$.

The pointers $\mathsf{dn}(\cdot)$ and $\mathsf{up}(\cdot)$ are defined such that $\mathsf{up}(u_i) = u_{i+1}$, $\mathsf{dn}(u_i) = u_{i-1}$ for all $1 \leq i \leq j-1$ and $\mathsf{up}(v_i) = v_{i+1}$, $\mathsf{dn}(v_i) = v_{i-1}$ for all $1 \leq i \leq k - 1$. Together with Conditions III.1 and III.2 this implies that all maxima q on a trail between a and b satisfy $f(\mathsf{dn}(q)) < f(q) < f(\mathsf{up}(q))$ and, except u_{j-1}, they satisfy $\mathsf{dn}(q) < q < \mathsf{up}(q)$ or $\mathsf{up}(q) < q < \mathsf{dn}(q)$. By Condition III.1 it holds that $\mathsf{dn}(u_{j-1}) = u_{j-2} > u_{j-1}$, and by the assumption that $a < b$ it holds that $b = \mathsf{up}(u_{j-1}) > u_{j-1}$. This shows that Invariant 3 is satisfied for all u_i with $1 \leq i \leq j-1$ and v_i with $1 \leq i \leq k-1$, i.e, for all nodes internal on a trail from a to b. Since all maxima except the special root are internal to some trail it follows that $\mathcal{B}$ satisfies Invariant 3.

We now prove that $\mathcal{B}$ satisfies Invariant 2. We need to show that for every minimum p it holds that $f(p) > f(\mathsf{low}(\mathsf{dth}(p)))$. Assume by contradiction that for some minimum p this is not the case and let $a = \mathsf{low}(\mathsf{dth}(p))$, i.e., we assume $f(a) = f(\mathsf{low}(\mathsf{dth}(p))) > f(p)$. The inequality $f(\mathsf{low}(\mathsf{dth}(p))) > f(p)$ implies that $\mathsf{dth}(p) = q$ is an ancestor of a for which a does not have the smallest value in the subtree of q. Condition II requires that there is a path $P(a, q)$ and this is a contradiction as $q \neq \mathsf{dth}(a)$. It follows that $f(p) > f(\mathsf{low}(\mathsf{dth}(p)))$ for every minimum p and that $\mathcal{B}$ satisfies Invariant 2.

It remains to show that the banana tree $\mathcal{B}$ also satisfies Invariant 1. Let q be some maximum except the special root, let Q be the path ending at q and let p be the leaf at the other end of Q. Assume $p < q$; the other case is symmetric. The path Q is contained entirely in a subtree S_1 of q and the construction of $\mathcal{B}$ is such that this subtree becomes the banana subtree rooted at q. By Condition I.1 either for all $s \in S_1 \colon s < q$ or for all $s \in S_1 \colon s > q$. Since $p \in S_1$, it follows that all $s \in S_1$ satisfy $s < q$. The subtree S_1 becomes the banana subtree rooted at q in $\mathcal{B}$ and thus the condition that $u < q$ for all u in the banana subtree rooted at q is satisfied. Let S_2 be the other subtree of q in T. By Condition I.1 for all $t \in S_2$ we have $t > q$. The node $\mathsf{dn}(q)$ in $\mathcal{B}$ is one of the nodes in S_2 and thus $\mathsf{dn}(q) > q$. By Conditions III.1 and III.2 all descendants v of $\mathsf{dn}(q)$ on the same trail as $\mathsf{dn}(q)$ satisfy $v > \mathsf{dn}(q)$ and thus $v > q$. All these nodes v are also nodes in S_2, since they are on the same path as $\mathsf{dn}(q)$ and have smaller value. The banana subtree of each v is obtained from the subtree of v in T and thus these banana subtrees consist of nodes in subtrees of S_2. It follows that every descendant t of $\mathsf{dn}(q)$ including $\mathsf{dn}(q)$ satisfies $t > q$, as required by Invariant 1. This concludes the proof that $\mathcal{B}$ satisfies Invariant 1. □

We now give an algorithm that turns a banana tree into path-decomposed binary tree.

Lemma A.7 (Merging Trails)**.** *There exists a deterministic algorithm to merge the trails of each banana in a banana tree satisfying Invariants 1 to 3, such that the resulting path-decomposed binary tree satisfies Conditions* I.1*,* I.2 *and* II*.*

Proof. Let $\mathcal{B}$ be a banana tree satisfying Invariants 1 to 3. Define the height of a banana subtree to be the longest simple path from the root of the banana subtree to a leaf following only $\mathsf{in}(\cdot)$, $\mathsf{mid}(\cdot)$ and $\mathsf{dn}(\cdot)$ pointers and with function value decreasing along the path. We show by induction over the height that any banana subtree of $\mathcal{B}$ satisfying Invariants 1 to 3 can be merged into a path-decomposed binary tree satisfying Conditions I.1, I.2 and II. Then, since the banana subtree rooted at the special root is equivalent to the banana tree $\mathcal{B}$, it follows that there is a deterministic algorithm to obtain a path-decomposed binary tree satisfying Conditions I.1, I.2 and II from $\mathcal{B}$.

We now give the algorithm for merging banana trees into path-decomposed binary trees. By induction over the height h we show that a banana subtree of $\mathcal{B}$ of height h with root q can be merged into a path-decomposed binary tree $T(q)$ such that Conditions I.1, I.2 and II are satisfied. To achieve this we also show that $\mathrm{Bth}(q) < q$ implies that q has no right child in $T(q)$ and $\mathrm{Bth}(q) > q$ implies that q has no left child in $T(q)$. This is needed to ensure that after assembling the subtrees of smaller height on a path, the nodes in the resulting binary tree are ordered correctly. Note that a banana tree consists of at least two nodes spanning a banana with empty trails, i.e., trails without interior nodes, so the base case is $h = 1$.

Base case: A banana tree of height 1 consists of a single banana spanned by a minimum p and a maximum q with $f(p) < f(q)$. We obtain a path-decomposed binary tree as follows: q becomes the tree with p as left child of q if $p < q$ and right child otherwise. There is a single path between p and q. This tree clearly satisfies Conditions I.1, I.2 and II. Furthermore, since $\mathrm{Bth}(q) = p$ if $p < q$, then q has no right child and if $p > q$ then q has no left child, so the claim holds.

Induction hypothesis: For some $h \geq 1$, for all $1 \leq j \leq h$ a banana subtree of $\mathcal{B}$ of height j can be merged into a path-decomposed binary tree T satisfying Conditions I.1, I.2 and II. Furthermore, $\mathrm{Bth}(q) < q$ implies that q has no right child in T and $\mathrm{Bth}(q) > q$ implies that q has no left child in T.

Induction step: Assume the induction hypothesis for some $h \geq 1$. We show that the claim holds for banana subtrees of height $h+1$. Let q be the root of a banana subtree of $\mathcal{B}$ of height $h+1$ and $p = \mathrm{Bth}(q)$. Banana subtrees rooted at a node on the banana spanned by p and q have height at most h, and by the induction hypothesis these banana subtrees can be merged into path-decomposed binary trees satisfying Conditions I.1, I.2 and II. Write $T(u)$ for the path-decomposed binary tree obtained from the banana subtree rooted at node u.

Let $s_0 = p, s_1, \ldots, s_\ell = q$ be the nodes on the trails between p and q ordered by function value. This ordering is unique since function values are distinct. Consider a node s_i for $1 \leq i \leq \ell - 1$. We show that for all $0 \leq j \leq i-1$, if $\mathrm{Bth}(s_i) < s_i$ then $s_i < s_j$ and $s_i < u$ for any descendant u of s_j, and if $\mathrm{Bth}(s_i) > s_i$ then $s_i > s_j$ and $s_i > u$ for any descendant u of s_j. Assume $\mathrm{Bth}(s_i) < s_j$; the other case is symmetric. By Invariant 1 we have $s_i < \mathsf{dn}(s_i)$ and for all descendants u of $\mathsf{dn}(s_i)$ it holds that $s_i < u$. Note that the nodes s_j for $0 \leq i-1$ that are on the same trail as s_i are descendants of $\mathsf{dn}(s_i)$ or $\mathsf{dn}(s_i)$. By Conditions III.1 and III.2 the nodes s_k for $k \leq i-1$ that are on the other trail than s_i satisfy $s_i < s_k$ and $\mathsf{dn}(s_k) < s_j$. For these nodes s_k Invariant 1 implies $s_k < \mathrm{Bth}(s_k)$ and all descendants v of s_k satisfy $s_k < v$. This proves that for all $0 \leq j \leq i-1$ both $s_i < s_j$ and $s_i < u$ for any descendant of u.

We iteratively assemble the path-decomposed binary trees rooted at the s_i for $1 \leq i \leq \ell - 1$ into a path-decomposed binary tree rooted at q by linking $T(s_i)$ to the tree assembled from the $T(s_j)$ for $j \leq i-1$.

We begin with $i = 1$. Note that $\mathsf{dn}(s_1) = s_0$, since s_1 must be the bottom-most node of the in- or mid-trail between p and b. By Invariant 1, if $\mathrm{Bth}(s_1) < s_1$ then $\mathsf{dn}(s_1) > s_1$ and if $\mathrm{Bth}(s_1) > s_1$ then $\mathsf{dn}(s_1) < s_1$. Assume $\mathrm{Bth}(s_1) < s_1$; the other case is symmetric. By the induction hypothesis the root s_1 of the tree $T(s_1)$ has no right child in this tree. We make $s_0 = \mathsf{dn}(s_1)$ the right child of s_1 and since $s_0 = \mathsf{dn}(s_1) > s_1$ this ensures that the resulting tree satisfies Condition I.1. By Invariant 3, $f(\mathsf{dn}(s_1)) < f(s_1)$, so Condition I.2 is also satisfied.

Now consider any i with $1 \leq i \leq \ell - 1$. In iteration $i-1$ we have already linked the $T(s_j)$ for $1 \leq j \leq i$ into a tree T_{i-1} satisfying Conditions I.1 and I.2. The process to combine T_{i-1} with $T(s_i)$ is similar to that for $i = 1$. By Invariant 1 we have either $\mathrm{Bth}(s_i) < s_i < \mathsf{dn}(s_i)$ or $\mathrm{Bth}(s_i) > s_i > \mathsf{dn}(s_i)$. Assume again that $\mathrm{Bth}(s_i) < s_i < \mathsf{dn}(s_i)$; the other case is symmetric. The nodes in T_{i-1} are exactly the nodes s_j for $j \leq i-1$ and their descendants, so as discussed above, for all $t \in T_{i-1}$ it holds that $\mathrm{Bth}(s_i) < s_i < t$. By the induction hypothesis s_i has no right child in $T(s_i)$. We can thus attach T_{i-1} as the right subtree of s_i in $T(s_i)$ and the resulting tree T_i satisfies Condition I.1. The tree T_i also satisfies Condition I.2, as T_{i-1} satisfies Condition I.2 and $f(s_i) > f(s_{i-1})$.

The algorithm finishes by making $T_{\ell-1}$ the left subtree of q if $p < q$ and the right subtree otherwise, which yields the tree $T(q)$. Furthermore, we make $p = s_0, s_1, \ldots, s_\ell = q$ a path in $T(q)$. Since all nodes u in $T_{\ell-1}$ are in the banana subtree rooted at q they satisfy $u < q$ if $p < q$ and $u > q$ if $p > q$. Thus $T(q)$ satisfies Condition I.1. Condition I.2 is also satisfied since $T_{\ell-1}$ satisfies this condition and $f(q) > f(s_{\ell-1})$. Note also that $T(q)$ has no right child if $p := \mathrm{Bth}(q) < q$ and no left child if $p := \mathrm{Bth}(q) > q$, since the banana subtree rooted at q satisfies Invariant 1.

It remains to show that $T(q)$ satisfies Condition II. There can be no minimum $a \neq p$ in the banana subtree rooted at q with $f(a) < f(p)$, as otherwise there would exist a minimum c in the banana subtree rooted at q violating $f(\mathsf{low}(\mathsf{dth}(c)) < f(c)$ required by Invariant 2. This implies that p has the smallest value in the banana subtree rooted at q. It is connected to the root q by a path, as required by Condition II. Since all $T(s_i)$ for $1 \leq i \leq \ell - 1$ satisfy Condition II by the induction hypothesis we now only need to show that the path ending at each s_i satisfies Condition II. Write t_i for the other end of the path ending at s_i. We need to show that s_i is the nearest ancestor to t_i such that s_i has a descendant with smaller value than t_i in $T(q)$. Since $f(p) < f(t_i)$, as discussed above, s_i is indeed such an ancestor, and since t_i is connected to the root of $T(s_i)$ by a path there exists no other such ancestor in $T(s_i)$. Thus, Condition II holds. This concludes the induction and the proof of the lemma. □

Finally, we state our result on uniqueness of banana trees.

Corollary A.8 (Invariants Determine the Banana Tree)**.** *Given a linear list of distinct values there is a unique banana tree satisfying Invariants 1 to 3.*

Proof. By lemma 4.2 there exists a unique path-decomposed binary tree for the linear list that satisfies Conditions $\mathsf{I}.1$, $\mathsf{I}.2$ and II. By lemmas A.6 and A.7 it follows that there is also a unique banana tree that satisfies Invariants 1 to 3. □

A.1 Slides

A *slide* occurs when an internal critical item becomes non-critical and its non-critical neighbor becomes critical. We distinguish two cases, based on whether the critical item is a maximum or a minimum.

Max Slide: Let p be a non-critical item, q a neighbor of p and a maximum. The value of p increases above the value of q or the value of q decreases below the value of p.

Min Slide: Let p be a non-critical item, q a neighbor of p and a minimum. The value of p decreases below the value of q or the value of q increases above the value of p.

In both cases, the number of critical items remains unchanged. If the order of critical items by function value is unaffected by the slide apart from q being replaced by p, then the banana tree can be updated by replacing the formerly critical item with the new critical item in the tree, which changes the label of the node associated with q to refer to p instead. We now prove that this correctly maintains the up-tree if the slide is caused by a sufficiently small change in value.

Lemma A.9 (Max-Slide)**.** *Let p and q be neighboring items. Let $\sigma \in [0,1)$ be such that in h_σ the item p is non-critical, q is a maximum and p, q are contiguous. Let $\theta \in (\sigma, 1]$ be such that in h_θ the item p is a maximum, q is non-critical and p, q are contiguous. The tree* $\mathrm{Up}(h_\theta)$ *is obtained from* $\mathrm{Up}(h_\sigma)$ *by replacing q with p.*

Proof. By lemma A.2, the order of maxima by function value is the same in h_σ and h_θ, with q replaced by p. The maps h_λ are defined such that they differ in exactly one fixed item j, so either $j = p$ or $j = q$. Neither p nor q is a minimum in h_σ or h_θ, so all minima have equal value in h_σ and h_θ. It then follows that the minimum s paired with q in h_σ is paired with p in h_θ. That is, if $W(s, q)$ is a window in h_σ, then $W(s, p)$ is a window in h_θ. Furthermore, $W(s, p)$ is nested into the same window in h_θ as $W(s, q)$ is in h_σ. No other window is affected by the change in value, since the order of critical items by function value is the same other than q being replaced by p. By lemma 4.1, s and q span a banana in h_σ and s and p span a banana in h_θ. Thus, replacing q with p in the up-tree for h_σ yields the up-tree for h_θ, as claimed. □

Lemma A.10 (Min-Slide). *Let p and q be neighboring items. Let $\sigma \in [0, 1)$ be such that in h_σ the item p is non-critical, q is a minimum and p and q are contiguous. Let $\theta \in (\sigma, 1]$ be such that in h_θ the item p is a minimum, q is non-critical and p and q are contiguous. The tree* $\mathrm{Up}(h_\theta)$ *is obtained from* $\mathrm{Up}(h_\sigma)$ *by replacing q with p.*

Proof. By lemma A.2, the order of minima by function value is the same in h_σ and h_θ, with q replaced by p. Since the values of maxima are equal in h_σ and h_θ it then follows that the maximum s paired with q in h_σ is paired with p in h_θ. That is, if $W(q, s)$ is a window in h_σ, then $W(p, s)$ is a window in h_θ. Furthermore, $W(p, s)$ is nested into the same window in h_θ as $W(s, q)$ is in h_σ. No other window is affected by the change in value, since the order of critical items by function value is the same other than q being replaced by p. By lemma 4.1, q, s span a banana in h_σ and p, s span a banana in h_θ. Thus, replacing q with p in the up-tree for h_σ yields the up-tree for h_θ, as claimed. □

A.2 Cancellations

Let p be a minimum, q be a maximum and a neighbor of p, such that there is a window $W(p, q)$ without any windows nested into it, i.e., such that p and q span a banana in the banana tree whose trails contain no other critical items. A cancellation occurs when the value of p increases above the value of q or the value of q decreases below the value of p. Then, both p and q become non-critical and the window spanned by p and q disappears. To update the up-tree we remove the banana spanned by p and q by connecting $\mathsf{up}(q)$ to $\mathsf{dn}(q)$ and discarding p and q.

Lemma A.11 (Cancellation). *Let p, q be neighboring items. Let $\sigma \in [0, 1)$ be such that in h_σ the item p is a minimum, q is a maximum and p and q are contiguous and paired in h_σ. Let $\theta \in (\sigma, 1]$ be such that in h_θ the items p and q are non-critical. Then nodes p and q span a banana in* $\mathrm{Up}(h_\sigma)$ *that does not contain any other critical items and removing this banana yields* $\mathrm{Up}(h_\theta)$.

Proof. Let $\gamma \in (\sigma, \theta]$ be such that in h_γ the items p and q are non-critical and contiguous. By lemma A.2, the order of items by function value is the same in h_σ and h_γ, except that p and q are swapped. In particular, the order of critical items by function value is the same in h_σ and h_γ. Since p, q are non-critical in h_γ and in h_θ it follows that the order of critical items remains the same. Thus, the order of critical items in h_σ and h_θ is identical.

Since p and q are contiguous in h_σ, there is no critical item with value between $h_\sigma(p)$ and $h_\sigma(q)$. As p and q are neighbors this implies that p and q are paired in h_σ: consider the process to obtain the pairing described in section 2.2; the component born at p in h_σ dies at q, since the component on the other side of q must be born at a lower value. Furthermore, there are no windows nested into $W(p, q)$, since there is no critical item with value in $[h_\sigma(p), h_\sigma(q)]$. All other windows in h_σ are also windows in h_θ, since the critical items have the same order by function value.

As $W(p, q)$ is a window of h_σ, by lemma 4.1 the nodes p and q span a banana in $\mathrm{Up}(h_\sigma)$. There are no windows nested into $W(p, q)$ and thus the trails between p and q are empty. Since p and q are non-critical in h_θ they are not present in $\mathrm{Up}(h_\theta)$. Thus, removing the banana spanned by p and q from $\mathrm{Up}(h_\sigma)$ yields $\mathrm{Up}(h_\theta)$, as claimed. □

A.3 Slides and Cancellations Involving Endpoints

When the criticality of the neighbor of an endpoint changes it can also change the endpoint from up-type to down-type or vice versa. Likewise, a change in the criticality of an endpoint leads to a change in the criticality of its neighbor. The operations needed to maintain the up-tree in these cases are similar to cancellations and slides. Let $q \in \{1, m\}$ be an endpoint and a be its neighbor. In the following we assume $a < q$, i.e., that q is the right endpoint of the interval. The case $q < a$ is symmetric. Let $0 \leq \sigma < \theta \leq 1$ such that $h_\sigma(q) > h_\sigma(a),\ h_\theta(q) < h_\theta(a)$ or $h_\sigma(q) < h_\sigma(a),\ h_\theta(q) > h_\theta(a)$. There are four cases:

1. $h_\sigma(q) > h_\sigma(a)$, $h_\theta(q) < h_\theta(a)$, a is a minimum in h_σ and non-critical in h_θ,
2. $h_\sigma(q) > h_\sigma(a)$, $h_\theta(q) < h_\theta(a)$, a is non-critical in h_σ and a maximum in h_θ,
3. $h_\sigma(q) < h_\sigma(a)$, $h_\theta(q) > h_\theta(a)$, a is a maximum in h_σ and non-critical in h_θ,
4. $h_\sigma(q) < h_\sigma(a)$, $h_\theta(q) > h_\theta(a)$, a is non-critical in h_σ and a minimum in h_θ.

We now give the algorithm for the case $a < q$.

```
cancel-or-slide-endpoint($a, q$):
  if $q$ changes from down-type to up-type then
    if $a$ becomes non-critical then
      remove $q$ and Bth($q$); replace $a$ by $q$
    else if $a$ becomes a maximum then
      let $p$ = Bth($q$); replace $q$ by $a$; replace $p$ by $q$
    endif
  else if $q$ changes from up-type to down-type then
    if $a$ becomes non-critical then
      replace $q$ by the hook; replace $a$ by $q$
    else if $a$ becomes a minimum then
      replace $q$ by $a$; insert $q$ as in($a$); add a banana between $q$ and its hook
    endif;
  endif.
```

Lemma A.12 (Changes Involving Endpoints)**.** *Let q be an endpoint, a be its neighbor and let $\check{q}$ be the hook neighboring q. Let $0 \leq \sigma < \theta \leq 1$ such that*

1. *either $h_\sigma(q) > h_\sigma(\check{q}) > h_\sigma(a)$, $h_\theta(q) < h_\theta(\check{q}) < h_\theta(a)$,*
2. *or $h_\sigma(q) < h_\sigma(\check{q}) < h_\sigma(a)$, $h_\theta(q) > h_\theta(\check{q}) > h_\theta(a)$,*

where q and $\check{q}$ are contiguous and $\check{q}$ and a are contiguous in h_σ and h_θ. Applying the algorithm ***cancel-or-slide-endpoint***(a, q) *to the up-tree* $\mathrm{Up}(h_\sigma)$ *yields the up-tree* $\mathrm{Up}(h_\theta)$*.*

Proof. Assume $a < q$; the other case is symmetric. We show how the change in function value from h_σ to h_θ affects the windows in each of the cases and that the algorithm correctly updates the bananas to match the windows of h_θ. Recall that the maps h_λ are defined such that they differ in a single fixed item. Thus, we can assume that either $h_\theta(q) = h_\sigma(q)$ or $h_\theta(a) = h_\sigma(a)$ and for all $u \neq q, a$ it holds that $h_\theta(u) = h_\sigma(u)$.

$h_\sigma(q) > h_\sigma(a)$, $h_\theta(q) < h_\theta(a)$, a is non-critical in h_θ (Line 4): There is a component in h_σ born at $\check{q}$ that dies at q and a component born at a that dies at an item b. The component born at $\check{q}$ merges into the component born at a. These components correspond to two windows $W(\check{q}, q)$ and $W(a, b)$, where the former is nested into the latter. In h_θ there are no components born at $\check{q}$ or a, as they are no longer homological critical points. Instead, there is a component born at q, which corresponds to a window $W(q, \cdot)$. As q, $\check{q}$ and $\check{q}$, a are contiguous in h_σ and h_θ the order of items by function value in h_σ and h_θ differs only in that q, $\check{q}$ and a are reversed by lemma A.2. Since q neighbors a it follows that the component born at q in h_θ dies at b, i.e., there is a window $W(q, b)$ in h_θ. The algorithm reflects this change in windows: the banana spanned by $\check{q}$ and q corresponding to $W(\check{q}, q)$ is removed; the banana spanned by a and b corresponding to $W(a, b)$ is replaced by a banana spanned by q and b corresponding to $W(q, b)$.

$h_\sigma(q) > h_\sigma(a)$, $h_\theta(q) < h_\theta(a)$, a is a maximum in h_θ (Line 6): Going from h_σ to h_θ can be seen as relabeling q such that it represents item a, relabeling $\check{q}$ such that it represents q and removing the item formerly associated with a. Changing the function values to reflect the values in h_θ does not affect the ordering of items, due to contiguity of q, $\check{q}$ and $\check{q}$, a. The corresponding change in the up-tree is to relabel the node representing q to represent item a, and relabeling the node representing the hook to represent item q, as the algorithm does in Line 6.

$h_\sigma(q) < h_\sigma(a)$, $h_\theta(q) > h_\theta(a)$, a is non-critical in h_θ (Line 10): Similarly to the previous case the change from h_σ to h_θ can be seen as insertion of an item $a^* < a$ next to a and contiguous with q in h_σ, then relabeling q to represent $\check{q}$, a to represent q and a^* to represent a. Adjusting the values of the items to reflect the values in h_θ does not affect the ordering of items, due to contiguity of q, $\check{q}$ and $\check{q}$, a. The corresponding change in the up-tree is to relabel the node representing q to represent $\check{q}$, and relabel the node representing a to represent q, as the algorithm does in Line 10.

$h_\sigma(q) < h_\sigma(a)$, $h_\theta(q) > h_\theta(a)$, a is a minimum in h_θ (Line 12): This is the reverse of the first case: there is a component born at q in h_θ and it is replaced by a component born at a and another component born at $\check{q}$ in h_θ. Reverse to the case above the window $W(q, b)$ is replaced by the window $W(a, b)$ and a new window $W(\check{q}, q)$ appears. The change from $W(q, b)$ to $W(a, b)$ is reflected by replacing q by a in the up-tree. The new window $W(\check{q}, q)$ must be nested into $W(a, b)$ as $h_\theta(a) < h_\theta(q) < h_\theta(b)$, since a, $\check{q}$ and q are contiguous in h_θ. Furthermore, $W(\check{q}, q)$ must be nested into the in-panel, as $b < a$. The nesting of $W(\check{q}, q)$ into the in-panel of $W(a, b)$ is reflected in the algorithm by adding a banana spanned by $\check{q}$ and q into the in-trail of the banana spanned by a and b.

The algorithm updates the bananas to reflect the windows and their nesting hierarchy in h_θ, which implies by lemma 4.1 that the new up-tree is $\mathrm{Up}(h_\theta)$. □

A.4 Anti-Cancellations

Let p and q be neighboring non-critical items. If the value of one changes such that both become critical, a new window is introduced. This is the reverse of a cancellation and we call it an *anti-cancellation*.

Denote by h_σ and h_θ the map before and after the anti-cancellation, respectively, and assume that in both h_σ and h_θ the items p and q are contiguous. Furthermore, assume that p becomes a minimum and q a maximum. To change $\mathrm{Up}(h_\sigma)$ into $\mathrm{Up}(h_\theta)$ we need to introduce a banana spanned by p and q. We recall the algorithm for the case $p < q$; the case $p > q$ is symmetric. The algorithm first identifies the maximum b closest to p such that $b < p < q$, i.e., such that p becomes the critical item next to b. It then walks down the path from b to its successor until it reaches the first node t such that $h_\theta(t) < h_\theta(q)$. The node q is then inserted into this path above t, and a new banana spanned by p and q is attached at q. To insert a node q between some nodes a and b, where a and b are neighbors on the same trail and $f(a) < f(b)$, we set $\mathsf{up}(q) = b$, $\mathsf{dn}(q) = a$ and make the pointer from a to b and the pointer from b to a point to q instead.

```
anticancel($p, q$):
    find the maximum $b$ closest to $p$ such that $b < p < q$
    if $b < \mathrm{Bth}(b)$ then $t = \mathsf{mid}(b)$ else $t = \mathsf{dn}(b)$ endif;
    while $h_\theta(t) > h_\theta(q)$ do $t = \mathsf{in}(t)$ endwhile;
    if $t$ is a leaf then if $t = \mathrm{Bth}(b)$ then insert $q$ between $t$ and $\mathsf{mid}(t)$
                                               else insert $q$ between $t$ and $\mathsf{in}(t)$
                          endif;
                     else insert $q$ between $t$ and $\mathsf{up}(t)$ into the trail of $t$
    endif.
    Set $\mathsf{in}(p) = \mathsf{mid}(p) = q$, $\mathsf{in}(q) = \mathsf{mid}(q) = p$, $\mathsf{dth}(p) = q$, $\mathsf{low}(q) = \mathsf{low}(\mathsf{dn}(q))$.
```

Lemma A.13 (Anti-Cancellation)**.** *Let p and q be neighboring items. Let $\sigma \in [0, 1)$ such that in h_σ both p and q are non-critical, $h_\sigma(p) > h_\sigma(q)$, and p and q are contiguous. Let $\theta \in (\sigma, 1]$ such that in h_θ the item p is a minimum, q is a maximum, and p and q are contiguous. Algorithm `anticancel`(p, q) applied to the up-tree $\mathrm{Up}(h_\sigma)$ yields the up-tree $\mathrm{Up}(h_\theta)$.*

Proof. We prove the lemma for the case where $p < q$; the other case is symmetric. The tree $\mathrm{Up}(h_\sigma)$ satisfies Invariants 1 to 3. We show that the tree obtained by `anticancel`also satisfies these invariants for h_θ, which by corollary A.8 implies that it is the unique up-tree for h_θ.

The algorithm first finds the maximum b closest to p such that $b < p < q$. Since p and q are non-critical in h_σ, the item p cannot be an endpoint. Furthermore, $h_\sigma(p) > h_\sigma(q)$ and thus this maximum b exists. After the while-loop terminates t satisfies $h_\theta(t) \leq h_\theta(q)$. Since p and q are contiguous in h_θ it holds that $h_\theta(t) \neq h_\theta(q)$ and thus $h_\theta(t) < h_\theta(q)$ and $h_\theta(t) < h_\theta(p)$.

The node q is inserted above t such that $\mathsf{dn}(q) = t$. As $\mathsf{low}(q)$ is set to $\mathsf{low}(\mathsf{dn}(q)) = \mathsf{low}(t)$, we have $h_\theta(\mathsf{low}(q)) < h_\theta(p)$ and it follows that $h_\theta(\mathsf{low}(q)) = h_\theta(\mathsf{low}(\mathsf{dth}(p))) < h_\theta(p)$. Since the $\mathsf{low}(\cdot)$ and $\mathsf{dth}(\cdot)$ pointers of all other nodes are unchanged between $\mathrm{Up}(h_\sigma)$ and the new up-tree, the new up-tree satisfies Invariant 2.

We now show that Invariant 1 holds in the new up-tree. Let t_0 be the node t as initialized in the first if-statement and t_i be the node t after the i-th iteration of the loop. Denote by I the last iteration, such that t_I is the node t after the loop terminates. Note that all t_i for $i \geq 0$ are present in $\mathrm{Up}(h_\sigma)$. As Invariant 1 holds for $\mathrm{Up}(h_\sigma)$, the initialization of t guarantees $b < t_0$. Furthermore, all t_i for $i \geq 1$ are descendants of t_0 and thus $b < t_i$ for all $i \geq 0$. By definition of b and since p and q are neighbors it also holds that $q < t_i$ for all $i \geq 0$ and thus $b < p < q < t_i$ for all $i \geq 0$.

To continue the proof of Invariant 1 we need the following claim:

Claim A.14. *The while-loop terminates. For all $0 \leq i \leq I - 1$ it holds that t_i is not a leaf, $\mathrm{Bth}(t_i) < t_i$ and $t_{i+1} < t_i$.*

Proof. There exists a minimum a such that $b < p < q < a$ and such that $h_\sigma(a) < h_\sigma(q)$ and such that there are no critical items u with $q < u < a$. This follows from a similar argument as for the existence of b: q is non-critical in h_σ and thus cannot be an endpoint and $h_\sigma(q) < h_\sigma(p)$, so the first critical item greater than q must be a minimum. Note that a must be either t_0 or a descendant of t_0 and that in the latter case it is the leftmost descendant of t_0 by Invariant 1. As the algorithm modifies the up-tree after the while-loop terminated, we only argue about $\mathrm{Up}(h_\sigma)$ in this proof. Recall that this up-tree satisfies Invariants 1 to 3 and that by lemma A.5 it also satisfies Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$.

By Invariant 1 $b = \mathsf{up}(t_0) < t_0$ and by Invariants 1 and 3 $\mathsf{in}(t_0) < t_0 < \mathsf{dn}(t_0)$, unless t_0 is a leaf. The while-loop assigns $t_1 = \mathsf{in}(t_0)$ and by Conditions $\mathsf{III}.1$, $\mathsf{III}.2$ and Invariant 1 this is the leftmost node among the other nodes on the banana with maximum and their descendants. Invariants 1 and 3 again $\mathsf{in}(t_1) < t_1$, unless t_1 is a leaf. This argument can be repeated for $\mathsf{in}(t_1), \mathsf{in}(\mathsf{in}(t_1)), \ldots$ until we reach a leaf. Thus, by following $\mathsf{in}(\cdot)$-pointers from t_0 we reach the leftmost leaf that is a descendant of t_0, which is a. As $h_\theta(a) = h_\sigma(a) < h_\sigma(q) \leq h_\theta(q)$, there exists a node $\mathsf{in}^*(t_0)$ with $h_\theta(\mathsf{in}^*(t_0)) < h_\theta(q)$ and hence the while-loop terminates. If the loop terminates in a leaf, then t_I is a leaf and all t_i with $0 \leq i \leq I-1$ are not leaves. If the loop terminates at an internal node, then all t_i for $0 \leq i \leq I$ are not leaves. We have also seen that $\mathsf{in}(t_i) < t_i$ for all $0 \leq i \leq I$ and this implies $\mathrm{Bth}(t_i) < t_i$ by Invariant 1. This proves the claim. ☐

We resume the proof of lemma A.13. Claim A.14 implies that the t_i are ordered such that $t_0 > t_1 > \cdots > t_I$. For all $0 \leq i \leq I-1$ the nodes p and q are inserted into the banana subtree rooted at t_i, as either $q = \mathsf{in}(t_i)$ or q is a descendant of $\mathsf{in}(t_i)$. Since $\mathrm{Bth}(t_i) < t_i$ and $p < q < t_i$ for $0 \leq i \leq I-1$, it follows that no t_i for $0 \leq i \leq I-1$ violates Invariant 1. Recall that q becomes $\mathsf{up}(t_I)$; this implies that neither the banana subtree rooted at t_I nor the descendants of $\mathsf{dn}(t_I)$ change and thus t_I satisfies the condition of Invariant 1 in the new tree. Finally, b also satisfies the condition of Invariant 1, as $b < p < q < t_0$ and p and q are inserted into the subtree of b that contains t_0. It follows that there is no node that violates the condition of Invariant 1 and thus the new up-tree satisfies Invariant 1.

We use the same claim to show that Invariant 3 is satisfied after inserting q and p. There are two cases: $t_I \neq t_0$ and $t_I = t_0$. In the first case, $q = \mathsf{in}(t_{I-1})$, $t_I = \mathsf{dn}(q)$ and $t_{I-1} = \mathsf{up}(q)$. As shown above $t_I > q$ and $t_{I-1} > q$. Furthermore, since the while-loop did not terminate for $t = t_{I-1}$ we have $h_\theta(q) < h_\theta(t_{I-1})$. It also holds that $h_\theta(t_I) < h_\theta(q)$, as the while-loop terminated for $t = t_I$. Thus, q satisfies the conditions in Invariant 3. In the case $t_I = t_0$, $q = \mathsf{up}(t_I)$, $\mathsf{up}(q) = b$ and either $q = \mathsf{mid}(b)$ or $q = \mathsf{dn}(b)$. We have $b < q < t_I$, i.e., $\mathsf{up}(q) < q < \mathsf{dn}(q)$. As in the other case $h_\theta(t_I) < h_\theta(q)$. It is also true that $h_\theta(q) < h_\theta(b)$, which follows from p and q being contiguous in h_θ and the definition of b. Thus, q again satisfies the conditions in Invariant 3. No other node can violate Invariant 3 and thus the new up-tree satisfies Invariant 3.

We have shown that the tree constructed by applying `anticancel`(p, q) to $\mathrm{Up}(h_\sigma)$ yields an up-tree that satisfies Invariants 1 to 3 for h_θ. By corollary A.8 this is the unique up-tree for h_θ. ☐

A.5 Interchanges

Interchanges of Maxima An interchange of maxima occurs when the value of a maximum j increases above the value of a maximum q or the value of q decreases below the value of j. This only has structural consequences on the up-tree if $q = \mathsf{up}(j)$, as this is the only case where one of the invariants, namely Invariant 3, is violated.

Denote by h_σ and h_θ the map before and after the interchange, such that $h_\sigma(j) < h_\sigma(q)$ and $h_\theta(j) > h_\theta(q)$. Assume $q < j$, which is the situation depicted in Figure 7. The case $j < q$ is symmetric. Let i and p be such that $j = \mathsf{dth}(i)$ and $q = \mathsf{dth}(p)$ in $\mathrm{Up}(h_\sigma)$. There are three cases: $j = \mathsf{dn}(q)$, $j = \mathsf{in}(q)$ and $j = \mathsf{mid}(q)$. If $j = \mathsf{in}(q)$ or $j = \mathsf{mid}(q)$, then $p = \mathsf{low}(j)$ and by Invariant 2

$h_\sigma(i) > h_\sigma(p)$. If $j = \mathsf{dn}(q)$ we further distinguish two cases depending on the order of $h_\sigma(i)$ and $h_\sigma(p)$. We now give the algorithm for processing an interchange of maximum j with maximum $q = \mathsf{up}(j)$ for the case $q < j$.

```
max-interchange($j, q$):
    Let $i = \mathrm{Bth}(j)$, $p = \mathrm{Bth}(q)$;
    if $j = \mathsf{in}(q)$ then
      remove $j$ from its trail; insert $j$ as $\mathsf{up}(q)$
    else if $j = \mathsf{mid}(q)$ then
      exchange $j$ and $q$;
      remove $q$ from its trail; insert $q$ as $\mathsf{dn}(j)$;
      swap in- and mid-trail of $i$ and $q$
    else if $j = \mathsf{dn}(q)$ then
      if $h_\sigma(i) < h_\sigma(p)$ then
        remove $q$ from its trail; insert $q$ as $\mathsf{in}(j)$
      else
        exchange $j$ and $q$;
        remove $q$ from its trail; insert $q$ as $\mathsf{mid}(j)$;
        swap in- and mid-trail of $i$ and $q$
      endif;
    endif.
```

The next lemma states that this algorithm correctly maintains the up-tree.

Lemma A.15 (Max-interchange)**.** *Let j and q be two items. Let $\sigma \in [0, 1)$ such that in h_σ the items j and q are maxima with $q = \mathsf{up}(j)$ and such that j and q are contiguous in h_σ. Let $\theta \in (\sigma, 1]$ such that in h_θ the items j and q are maxima with $h_\theta(j) > h_\theta(q)$ and such that j and q are contiguous in h_θ. Applying `max-interchange`(j, q) to the up-tree $\mathrm{Up}(h_\sigma)$ yields the up-tree $\mathrm{Up}(h_\theta)$.*

Proof. The tree $\mathrm{Up}(h_\sigma)$ satisfies Invariants 1 to 3. Recall that by lemma A.5 this up-tree also satisfies Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$. We show that the tree obtained by `max-interchange`also satisfies these invariants for h_θ, which by corollary A.8 implies that it is the unique up-tree for h_θ.

Write I_q for the nodes internal to the in-trail beginning at q and M_q for the nodes internal to the mid-trail beginning at q in $\mathrm{Up}(h_\sigma)$; similarly, write I_p, I_i, I_j for the nodes internal to the in-trails and M_p, M_i, M_j for the nodes internal to the mid-trails at p, i, j.

There are three cases in `max-interchange`: $j = \mathsf{in}(q)$, $j = \mathsf{mid}(q)$ and $j = \mathsf{dn}(q)$. As $q = \mathsf{up}(j)$ there is no other case to consider. The case $j = \mathsf{dn}(q)$ is further divided into two cases based on the values of $\mathrm{Bth}(j)$ and $\mathrm{Bth}(q)$. We describe for each case how the trails change from $\mathrm{Up}(h_\sigma)$ to the new tree and show that Invariants 1 and 3 hold in the new tree. Afterwards we prove Invariant 2.

Case 1 **($j = \mathsf{in}(q)$):** Assume $j < q$; the other case is symmetric. Note that $i = \mathrm{Bth}(j) < j$ and $p = \mathrm{Bth}(q) < q$. The in-trail and mid-trail between i and j remain the in-trail and mid-trail between i and j, and the in-trail and mid-trail between p and q remain the in-trail and mid-trail between p and q, with the exception that j is removed from the in-trail between p and q. In the new tree the nodes internal to the in-trail between i and j are thus I_j and the nodes internal to the mid-trail between i and j are M_j. Similarly, the nodes internal to the in-trail between p and q are $I_q \setminus \{j\}$ and the nodes internal to the mid-trail between p and q are M_q.

To see that Invariant 1 holds we analyze the subtrees of j and q to show that they satisfy the condition of Invariant 1 in the new up-tree. For any other node the condition holds in the new tree as the nodes in the respective subtrees do not change. First, consider the node q. The node $\mathsf{dn}(q)$ is the same in $\mathrm{Up}(h_\sigma)$ and in the new tree. The algorithm also does not modify the descendants of $\mathsf{dn}(q)$. Thus, for all descendants u of $\mathsf{dn}(q)$ including $\mathsf{dn}(q)$ we have $q < u$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$, as $p = \mathrm{Bth}(q) < q$. The banana subtree rooted at q in the new tree differs from that in $\mathrm{Up}(h_\sigma)$ only by the banana subtree rooted at j, which the algorithm removes. Thus, for all nodes v in the banana subtree rooted at q it holds that $v < q$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$, as $p = \mathrm{Bth}(q) < q$. It follows that q satisfies the condition for Invariant 1. Now consider the node j. The algorithm does not change the banana subtree rooted at j and thus for all v in this subtree we have $v < j$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$, as $i = \mathrm{Bth}(j) < j$. The other subtree of j, i.e., the descendants of $\mathsf{dn}(j)$ differ significantly: in $\mathrm{Up}(h_\sigma)$ these were the descendants of nodes on the in-trail below j; in the new tree they are the descendants of q. For the descendants u_1 of $\mathsf{dn}(q)$ it is easy to see that $j < u_1$, as $j < q < u_1$, which follows from the discussion for q and the assumption that $j < q$. For the descendants u_2 of nodes on the banana spanned by p and q in the new tree it follows from Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ and Invariant 1 that $j < u_2$: the node j is the topmost node on the in-trail, and with $j < q$ this implies that the other maxima t on the banana satisfy $j < t$; nodes u_3 in the banana subtrees rooted at each t are either descendants of $\mathsf{dn}(j)$, which implies $j < u_3$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$, or t is on the mid-trail between p and q, which by $p < q$ and Invariant 1 also implies $j < t < u_3$. It follows that all descendants u of $\mathsf{dn}(j) = q$, including $\mathsf{dn}(j)$ satisfy $j < u$. Thus, the condition of Invariant 1 holds for j.

We now prove that Invariant 3 holds by again analyzing j and q; for the remaining nodes the conditions of Invariant 3 follow directly from Invariant 3 for $\mathrm{Up}(h_\sigma)$. Let $u_q = \mathsf{up}(q)$ and $d_q = \mathsf{dn}(q)$ in $\mathrm{Up}(h_\sigma)$. In the new tree $d_q = \mathsf{dn}(q)$, $u_q = \mathsf{up}(j)$ and $j = \mathsf{up}(q)$. By the assumption $j < q$ and Invariant 3 for $\mathrm{Up}(h_\sigma)$ we have $j < q < d_q$ in the new tree, i.e., $\mathsf{up}(q) < q < \mathsf{dn}(q)$. If $q = \mathsf{in}(u_q)$ in $\mathrm{Up}(h_\sigma)$ then $q < u_q$ and in the new tree $j = \mathsf{in}(u_q)$ and $j < u_q$. Thus, if $j = \mathsf{in}(\mathsf{up}(j)) = \mathsf{in}(u_q)$ in the new tree then $j < \mathsf{up}(j)$ and $j < \mathsf{dn}(j) = q$. Otherwise, if $q \neq \mathsf{in}(u_q)$ in $\mathrm{Up}(h_\sigma)$ then $u_q < q$ and $u_q < j$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$. In the new tree this implies $j \neq \mathsf{in}(\mathsf{up}(j)) = \mathsf{in}(u_q)$ and $u_q = \mathsf{up}(j) < j < \mathsf{dn}(j) = q$. The inequalities $h_\theta(\mathsf{up}(j)) > h_\theta(j) > h_\theta(\mathsf{dn}(j)) = h_\theta(q)$ and $h_\theta(j) = h_\theta(\mathsf{up}(q)) > h_\theta(q) > h_\theta(\mathsf{dn}(q))$ follow directly from the fact that j and q are contiguous in h_θ and from Invariant 3 for $\mathrm{Up}(h_\sigma)$. Thus, j and q satisfy the conditions of Invariant 3 in the new tree and it follows that Invariant 3 holds in the new tree.

Case 2 ($j = \mathsf{mid}(q)$): Assume $j < q$; the other case is symmetric. Note that $i = \mathrm{Bth}(j) > j$ and $p = \mathrm{Bth}(q) < q$. The nodes q and j exchange their partners, i.e., in the new up-tree $\mathrm{Bth}(q) = i$ and $\mathrm{Bth}(j) = p$. The in-trail and mid-trail between i and j become the mid- and in-trail between i and q (notice that the trails change from in to mid and vice versa); the in-trail and mid-trail between p and q become the in- and mid-trail between p and q, with j removed from the mid-trail. That is, in the new tree the nodes internal to the in-trail between i and q are the nodes M_j and the nodes internal to the mid-trail between i and q are the nodes I_j. The nodes internal to the in-trail between p and j are the nodes I_q and the nodes internal to the mid-trail between p and j are the nodes $M_q \setminus \{j\}$.

The proof of Invariant 3 is identical to that of Case 1. To see that Invariant 1 holds we again analyze the subtrees of j and q to show that they satisfy the condition of Invariant 1. As in the previous case the remaining nodes satisfy the condition in the new tree, as their subtrees are unchanged by the algorithm. For the node q observe that $\mathsf{dn}(q)$ and its descendants are the same in $\mathrm{Up}(h_\sigma)$ and the new tree. Furthermore, the nodes in the banana subtree rooted at q in the new tree are also in the banana subtree rooted at q in $\mathrm{Up}(h_\sigma)$. The condition of Invariant 1 thus holds for q by Invariant 1 for $\mathrm{Up}(h_\sigma)$. Now we consider the node j. The descendants of $\mathsf{dn}(j)$ are (1) the descendants of $\mathsf{dn}(q)$, including $\mathsf{dn}(q)$, and (2) the node q along with the banana subtree rooted at q. For the descendants u_1 of $\mathsf{dn}(q)$, including $\mathsf{dn}(q)$, it holds that $j < u_1$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$. The banana subtree rooted at q in the new tree is the banana subtree rooted at j in $\mathrm{Up}(h_\sigma)$, and for nodes u_2 in this banana subtree $j < u_2$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$, since $j < i = \mathrm{Bth}(j)$. With the assumption $j < q$ it follows that the descendants u of $\mathsf{dn}(j)$ including $\mathsf{dn}(j)$ satisfy $j < u$ in the new tree. The nodes v in the banana subtree j in the new tree are descendants of maxima $t \neq j$ on a trail between p and q. By Conditions III.1, III.2 and Invariant 1 for $\mathrm{Up}(h_\sigma)$ these nodes satisfy $v < j$: the node j is the topmost node of the right trail of the banana spanned by p and q; the nodes t_1 on the left trail satisfy $t_1 < j$ by Conditions III.1 and III.2 and the nodes v_1 in the banana subtrees rooted at t_1 satisfy $v_1 < t_1 < j$ by Invariant 1; the nodes t_2 on the right trail below j are descendants of $\mathsf{dn}(j)$, and they along with their descendants v_2 satisfy $t_2 < j$ and $v_2 < j$ by Invariant 1. It follows that j satisfies the condition of Invariant 1. Since j and q satisfy the condition of Invariant 1 it follows that Invariant 1 holds in the new tree.

Case 3.1 ($j = \mathsf{dn}(q)$ and $h_\sigma(i) < h_\sigma(p)$): Assume $q < j$; the other case is symmetric. Note that $i = \mathrm{Bth}(j) < j$ and $p = \mathrm{Bth}(q) < q$. The in-trails and mid-trails remain the same, with the exception that q is added to the in-trail between i and j. That is, in the new tree the nodes internal to the in-trail between i and j are $I_j \cup \{q\}$, the nodes internal to the mid-trail between i and j are M_j, the nodes internal to the in-trail between p and q are I_q and the nodes internal to the mid-trail between p and q are M_q.

To see that Invariant 1 holds we again analyze the subtrees of j and q, as in the previous cases. The banana subtree rooted at q is the same in $\mathrm{Up}(h_\sigma)$ and the new tree. The set of descendants of $\mathsf{dn}(q)$ including $\mathsf{dn}(q)$ in the new tree is a subset of that in $\mathrm{Up}(h_\sigma)$. Thus, the condition of Invariant 1 holds for q in the new tree by Invariant 1 for $\mathrm{Up}(h_\sigma)$. For the node j the $\mathsf{dn}(j)$ and its descendants do not change. The banana subtree rooted at j in the new tree is equal to that in $\mathrm{Up}(h_\sigma)$, with the addition of q and the banana subtree rooted at q. By assumption $q < j$ and by Invariant 1 for $\mathrm{Up}(h_\sigma)$ the nodes v_1 in the banana subtree rooted at q satisfy $v_1 < q$ and thus $v_1 < j$. The remaining nodes v_2 in the banana subtree of j satisfy $v_2 < j$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$. Thus, for all nodes v in the banana subtree rooted at j we have $v < j$. It follows that j satisfies the condition of Invariant 1 and that the new up-tree satisfies Invariant 1.

We now show that Invariant 3 holds. Let $i_j = \mathsf{in}(j)$, $u_q = \mathsf{up}(q)$ in $\mathrm{Up}(h_\sigma)$. Observe that q is inserted at the top of the left trail between i and j, i.e., $q = \mathsf{in}(j)$ in the new tree. By Invariant 1 and the assumption that $q < j$ it follows that $q < \mathsf{up}(q) = j$ and $q < \mathsf{dn}(q)$. Note that $u_q = \mathsf{up}(j)$ in the new tree. The inequality $j < \mathsf{dn}(j)$ holds in $\mathrm{Up}(h_\sigma)$ since j is on a left trail, and holds in the new tree since $\mathsf{dn}(j)$ does not change. If $q = \mathsf{in}(u_q)$ in $\mathrm{Up}(h_\sigma)$, then $q < u_q$ and by Invariant 1 $j < u_q$. In the new tree $j = \mathsf{in}(u_q)$ and thus $j < \mathsf{up}(j) = u_q$ and $j < \mathsf{dn}(j)$. Otherwise, if $q \neq \mathsf{in}(u_q)$ in $\mathrm{Up}(h_\sigma)$, then $u_q < q < j$. Thus, in the new tree $u_q = \mathsf{up}(j) < j < \mathsf{dn}(j)$. The inequalities $h_\theta(\mathsf{up}(q)) > h_\theta(q) > h_\theta(\mathsf{dn}(q))$ and $h_\theta(\mathsf{up}(j)) > h_\theta(j) > h_\theta(\mathsf{dn}(j))$ follow directly from Invariant 3 for $\mathrm{Up}(h_\sigma)$ and the fact that j and q are contiguous in h_θ.

Case 3.2 ($j = \mathsf{dn}(q)$ and $h_\sigma(i) > h_\sigma(p)$): Assume $q < j$; the other case is symmetric. Note that $i = \mathrm{Bth}(j) < j$ and $p = \mathrm{Bth}(q) < q$. The nodes q and j exchange their partners, i.e., in the new up-tree $\mathrm{Bth}(q) = i$ and $\mathrm{Bth}(j) = p$. The in-trail and mid-trail between i and j become the mid-trail and in-trail between i and p (notice that the trails change from in to mid and vice versa); the in-trail and mid-trail between p and q become the in-trail and mid-trail between p and j. Note that q is added to the mid-trail between p and j. Thus, in the new tree the nodes internal to the in-trail between p and j are I_q, the nodes internal to the mid-trail between p and j are $M_q \cup \{q\}$, the nodes internal to the in-trail between i and q are M_j and the nodes internal to the mid-trail between i and q are I_q.

The proof for Invariant 3 is identical to that of Case 3.1. The proof for Invariant 1 is similar to that of Case 3.1 and we point out the differences. The banana subtree rooted at q in the new tree is the banana subtree rooted at j in $\mathrm{Up}(h_\sigma)$. Since the nodes u in this subtree were descendants of $\mathsf{dn}(q)$ in $\mathrm{Up}(h_\sigma)$ it follows from Invariant 1 for $\mathrm{Up}(h_\sigma)$ that $q < u$. The descendants of $\mathsf{dn}(q)$ including $\mathsf{dn}(q)$ are nodes formerly in the banana subtree rooted at q, and these nodes v satisfy $v < q$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$. As in the previous case, the descendants of $\mathsf{dn}(j)$ including $\mathsf{dn}(j)$ are unchanged. The banana subtree rooted at j in the new tree is the banana subtree rooted at j in $\mathrm{Up}(h_\sigma)$ combined with the banana subtree rooted at q in $\mathrm{Up}(h_\sigma)$. Invariant 1 follows by the same argument as above.

It remains to prove that Invariant 2 holds in the new tree. Observe that $\mathsf{low}(\cdot)$ pointers only change for q and j. Consequently, we need to show that p and i satisfy $h_\theta(\mathsf{low}(\mathsf{dth}(p)) < h_\theta(p)$ and $h_\theta(\mathsf{low}(\mathsf{dth}(i))) < h_\theta(i)$. We consider the three cases above:

Case 1 ($j = \mathsf{in}(q)$): We have $\mathsf{low}(j) = p$ in $\mathrm{Up}(h_\sigma)$ and $\mathsf{low}(j) = \mathsf{low}(q)$ in the new tree. Recall that $j = \mathsf{dth}(i)$ and $q = \mathsf{dth}(p)$ in both $\mathrm{Up}(h_\sigma)$ and in the new tree. Since $\mathrm{Up}(h_\sigma)$ satisfies Invariant 2, we know that $h_\sigma(i) > h_\sigma(\mathsf{low}(j)) = h_\sigma(p) > h_\sigma(\mathsf{low}(q))$. As $h_\sigma(u) = h_\theta(u)$ for $u \neq j, q$, it holds that $h_\theta(i) > h_\theta(\mathsf{low}(j)) = h_\theta(\mathsf{low}(q))$ and $h_\theta(p) > h_\theta(\mathsf{low}(q))$.

Case 2 ($j = \mathsf{mid}(q)$): We have $\mathsf{low}(j) = p$ in $\mathrm{Up}(h_\sigma)$ and $\mathsf{low}(j) = \mathsf{low}(q)$ in the new tree. In the new tree $\mathsf{dth}(i) = q$ and $\mathsf{dth}(p) = j$. It holds that $h_\sigma(i) > h_\sigma(p) > h_\sigma(\mathsf{low}(q))$ by Invariant 2 for $\mathrm{Up}(h_\sigma)$ and thus $h_\theta(i) > h_\theta(\mathsf{low}(q))$. Similarly $h_\theta(p) = h_\sigma(p) > h_\sigma(\mathsf{low}(q)) = h_\theta(\mathsf{low}(q)) = h_\theta(\mathsf{low}(j))$, i.e., $h_\theta(p) > h_\theta(\mathsf{low}(j))$.

Case 3 ($j = \mathsf{dn}(q)$): We distinguish the two cases $h_\sigma(i) < h_\sigma(p)$ and $h_\sigma(i) > h_\sigma(p)$.

$h_\sigma(i) < h_\sigma(p)$**:** We have $\mathsf{low}(q) = \mathsf{low}(j)$ in $\mathrm{Up}(h_\sigma)$ and $\mathsf{low}(j) \neq \mathsf{low}(q) = i$ in the new tree. Note that $\mathsf{low}(j)$ is the same in both trees. In the new tree it holds that $h_\theta(p) = h_\sigma(p) > h_\sigma(\mathsf{low}(q)) = h_\sigma(i) = h_\theta(i)$, i.e., $h_\theta(p) > h_\theta(i)$. As $\mathsf{low}(j)$ does not change and $\mathsf{dth}(i) = j$ in both trees, $h_\theta(i) > h_\theta(\mathsf{low}(j))$ in the new tree.

$h_\sigma(i) > h_\sigma(p)$**:** We have $\mathsf{low}(q) = \mathsf{low}(j)$ in $\mathrm{Up}(h_\sigma)$ and $\mathsf{low}(j) \neq \mathsf{low}(q) = p$ in the new tree. In the new tree we also have $\mathsf{dth}(i) = q$ and $\mathsf{dth}(p) = j$. Note again that $\mathsf{low}(j)$ is the same in both trees. In the new tree it holds that $h_\theta(p) = h_\sigma(p) > h_\sigma(\mathsf{low}(j)) = h_\theta(\mathsf{low}(j))$, since in $\mathrm{Up}(h_\sigma)$ $h_\sigma(p) > h_\sigma(\mathsf{low}(q))$ and $\mathsf{low}(j) = \mathsf{low}(q)$. Furthermore, in the new tree $h_\theta(i) = h_\sigma(i) > h_\sigma(p) = h_\theta(p)$.

In all cases p and i satisfy the required condition and thus Invariant 2 is satisfied in the new tree. We have shown that the new up-tree obtained by applying `max-interchange`to $\mathrm{Up}(h_\sigma)$ satisfies Invariants 1 to 3, which implies that it is indeed the unique up-tree for h_θ. □

Interchanges of Minima Let i be a minimum and $p = \mathsf{low}(\mathsf{dth}(i))$. In an up-tree satisfying Invariant 2 the function value of i is greater than the function value of p. If this relation changes such that the function value of p becomes greater than the function value of i, Invariant 2 is violated. To fix this violation we perform an *interchange of minima*.

Let $\sigma \in [0, 1)$ such that $h_\sigma(i) > h_\sigma(\mathsf{low}(\mathsf{dth}(i))) = h_\sigma(p)$ and such that i and p are contiguous. Let $\theta \in (\sigma, 1]$ such that $h_\theta(i) < h_\theta(p)$ and such that i and p are contiguous. Write $j = \mathsf{dth}(i)$ and $q = \mathsf{dth}(p)$. We give the detailed algorithm for the case $q < p$; the other case is symmetric.

```
min-interchange($i, p$):
  if $\mathsf{low}(\mathsf{dth}(i)) \neq p$ then exit endif.
  Let $j = \mathsf{dth}(i)$, $q = \mathsf{dth}(p)$.
  Let $u_j = \mathsf{up}(j)$, $d_j = \mathsf{dn}(j)$, $i_j = \mathsf{in}(j)$, $m_j = \mathsf{mid}(j)$.
  From $q$ down along the trail not containing $j$ find
    1. first node $s^-$ such that $h_\sigma(s^-) < h_\sigma(j)$
    2. last node $s^+$ such that $h_\sigma(s^+) > h_\sigma(j)$.
  Set $\mathsf{dn}(j) \leftarrow m_j$, $\mathsf{mid}(j) \leftarrow d_j$, $\mathsf{in}(j) \leftarrow s^-$, $\mathsf{up}(j) \leftarrow s^+$.
  if $q < j < p$ then
    Swap $\mathsf{in}(i)$ and $\mathsf{mid}(i)$;
    if $i_j = i$ then $\mathsf{mid}(i_j) = u_j$ else $\mathsf{up}(i_j) = u_j$ endif;
    if $u_j = q$ then $\mathsf{mid}(u_j) = i_j$ else $\mathsf{dn}(u_j) = i_j$ endif;
    if $s^+ = q$ then $\mathsf{in}(s^+) = j$ else $\mathsf{dn}(s^+) = j$ endif;
    if $s^- = p$ then $\mathsf{in}(s^-) = j$ else $\mathsf{up}(s^-) = j$ endif
  else if $q < p < j$
    Swap $\mathsf{in}(p)$ and $\mathsf{mid}(p)$;
    if $i_j = i$ then $\mathsf{in}(i_j) = u_j$ else $\mathsf{up}(i_j) = u_j$ endif;
    if $u_j = q$ then $\mathsf{in}(u_j) = i_j$ else $\mathsf{dn}(u_j) = i_j$ endif;
    if $s^+ = q$ then $\mathsf{mid}(s^+) = j$ else $\mathsf{dn}(s^+) = j$ endif;
    if $s^- = p$ then $\mathsf{in}(s^-) = j$ else $\mathsf{up}(s^-) = j$ endif
  endif.
  Set $\mathsf{dth}(i) = q$, $\mathsf{dth}(p) = j$.
  Set $\mathsf{low}(u) = i$ for nodes $u \neq q$ between $u_j$ and $q$ and $j$ and $q$, including $u_j$ and $j$.
```

Lemma A.16 (Min-interchange)**.** *Let p and i be items. Let $\sigma \in [0, 1)$ such that p and i are minima and contiguous in h_σ, $h_\sigma(i) > h_\sigma(p)$ and $p = \mathsf{low}(\mathsf{dth}(i))$ in $\mathrm{Up}(h_\sigma)$. Let $\theta \in (\sigma, 1]$ such that p and i are minima and contiguous in h_θ, and $h_\theta(i) < h_\theta(p)$. Applying `min-interchange`$(i, p)$ to $\mathrm{Up}(h_\sigma)$ yields the up-tree for h_θ.*

Proof. The algorithm defines $j = \mathsf{dth}(i)$, $q = \mathsf{dth}(p)$. Assume $q < p$; the other case is symmetric. We show that by applying `min-interchange`to $\mathrm{Up}(h_\sigma)$, which satisfies Invariants 1 to 3, we obtain an up-tree satisfying Invariants 1 to 3 for h_θ. By corollary A.8 this is the unique up-tree for h_θ, $\mathrm{Up}(h_\theta)$. Note that by definition of h_λ the maps h_σ and h_θ differ in either the value of i or the value of p, and are equal in the values of all other items.

In Line 5 finds two nodes on the trail between p and q that does not contain j: s^- with $h_\sigma(s^-) < h_\sigma(j)$ and s^+ with $h_\sigma(s^+) > h_\sigma(j)$, with s^- being the highest such node and s^+ being the lowest such node along the trail. Since by Invariant 3 nodes are ordered by function value along trails, s^- and s^+ are neighbors along the trail.

It may be the case that s^- is a node internal to the trail or that $s^- = p$, and similarly s^+ is either internal to the trail or $s^+ = q$. Similar statements hold for the nodes defined in Line 4:

- $u_j = \mathsf{up}(j)$ is either internal to the trail between p and q or $u_j = q$,

- $d_j = \mathsf{dn}(j)$ is either internal to the trail between p and q or $d_j = p$,
- $i_j = \mathsf{in}(j)$ is either internal to the in-trail between i and j or $i_j = i$,
- $m_j = \mathsf{mid}(j)$ is either internal to the mid-trail between i and j or $m_j = i$.

The if-statement in Line 9 distinguishes two cases: $q < j < p$ and $q < p < j$. The cases correspond to j being on the mid-trail between p and q and j being on the in-trail between p and q. As $j = \mathsf{dth}(i)$ and $p = \mathsf{low}(\mathsf{dth}(i))$ the node j must be in one of the trails between p and q. Thus, these two cases are the only cases that can occur.

We first prove Invariant 3. Recall that by lemma A.5, $\mathrm{Up}(h_\sigma)$ satisfies Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$. We consider each node for which the $\mathsf{up}(\cdot)$ or $\mathsf{dn}(\cdot)$ pointer changes individually.

Node j**:** in the new tree $\mathsf{up}(j) = s^+$ and $\mathsf{dn}(j) = m_j$. By definition of h_λ it holds that $h_\sigma(s^+) = h_\theta(s^+)$ and $h_\sigma(j) = h_\theta(j)$. As $h_\sigma(s^+) > h_\sigma(j)$ it follows that $h_\theta(\mathsf{up}(j)) > h_\theta(j)$. If $m_j \neq i$ then $h_\sigma(m_j) = h_\theta(m_j)$; if $m_j = i$ then $h_\sigma(m_j) \geq h_\theta(m_j)$ as either $h_\sigma(i) = h_\theta(i)$ or $h_\theta(i) < h_\theta(p) = h_\sigma(p)$. Thus, $h_\theta(m_j) < h_\theta(j)$. It follows that $h_\theta(\mathsf{dn}(j)) < h_\theta(j) < h_\theta(\mathsf{up}(j))$. For the other conditions of Invariant 3 we consider the two cases of Line 9 separately:

$q < j < p$**:** the node j is on the mid-trail between p and q, which by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ implies that $j < \mathsf{dn}(j)$ in $\mathrm{Up}(h_\sigma)$. It follows that $m_j < j$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$, since in that tree m_j is in the banana tree rooted at j. Thus, in the new tree $m_j = \mathsf{dn}(j) < j$. If $s^+ = q$, then $q = s^+ = \mathsf{up}(j) < j$. Since in this case $j = \mathsf{in}(s^+)$ (see Line 13) in the new tree j satisfies point 3 of Invariant 3. If $s^+ \neq q$, then $j < s^+$ by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$, as s^+ is in the in-trail between p and q. Thus $\mathsf{dn}(j) < j < \mathsf{up}(j)$ in the new tree and hence j satisfies point 2 of Invariant 3.

$q < p < j$**:** the node j is on the in-trail between p and q, which by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ implies that $\mathsf{dn}(j) < j$ in $\mathrm{Up}(h_\sigma)$. It follows that $j < m_j$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$. Thus, in the new tree $j < \mathsf{dn}(j) = m_j$. If $s^+ = q$, then $q = s^+ = \mathsf{up}(j) < j$. Since in this case $j = \mathsf{mid}(s^+)$ (see Line 19) in the new tree j satisfies point 2 of Invariant 3. If $s^+ \neq q$, then $s^+ < j$ by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$, as s^+ is in the mid-trail between p and q. Thus, $\mathsf{up}(j) < j < \mathsf{dn}(j)$ in the new tree and hence j satisfies point 2 of Invariant 3.

Node i_j**:** we only need to consider the case where $i_j \neq i$, as i is a minimum and does not affect Invariant 3. Since $\mathsf{dn}(i_j)$ is the same in $\mathrm{Up}(h_\sigma)$ and in the new tree, by Invariant 3 it holds that $h_\sigma(\mathsf{dn}(i_j)) < h_\sigma(i_j)$. If $\mathsf{dn}(i_j) \neq i$ then $h_\sigma(\mathsf{dn}(i_j)) = h_\theta(\mathsf{dn}(i_j))$; otherwise if $\mathsf{dn}(i_j) = i$ then $h_\theta(\mathsf{dn}(i_j)) \leq h_\sigma(\mathsf{dn}(i_j))$ by the assumptions on i and p. Hence, we have $h_\theta(\mathsf{dn}(i_j)) < h_\sigma(i_j) = h_\theta(i_j)$. In the new tree $\mathsf{up}(i_j) = u_j$. By Invariant 3 for $\mathrm{Up}(h_\sigma)$ and the definition of h_λ it holds that $h_\theta(u_j) = h_\sigma(u_j) > h_\sigma(j) > h_\sigma(i_j) = h_\theta(i_j)$. Thus, $h_\theta(\mathsf{dn}(i_j)) < h_\theta(i_j) < h_\theta(\mathsf{up}(i_j))$. For the other conditions of Invariant 3 we consider the two cases of Line 9 separately:

$q < j < p$**:** j is on the mid-trail and thus by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ $j < \mathsf{dn}(j)$ in $\mathrm{Up}(h_\sigma)$. By Invariant 1 it follows that in $\mathrm{Up}(h_\sigma)$ $i_j = \mathsf{in}(j) < j$ and by Invariant 3 $i_j < \mathsf{dn}(i_j)$. Also by Invariant 3 $u_j = \mathsf{up}(j) < j$ in $\mathrm{Up}(h_\sigma)$. As i_j and j are in the same subtree of u_j in $\mathrm{Up}(h_\sigma)$ Invariant 1 implies that $u_j < i_j$ since $u_j < j$. Thus, in the new tree $u_j = \mathsf{up}(i_j) < i_j < \mathsf{dn}(i_j)$.

$q < p < j$**:** j is on the in-trail and thus by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ $\mathsf{dn}(j) < j$ in $\mathrm{Up}(h_\sigma)$. By Invariant 1 and Invariant 3 it follows that in $\mathrm{Up}(h_\sigma)$ $j = \mathsf{up}(i_j) < i_j$ and $\mathsf{dn}(i_j) < i_j$.

If $u_j = q$ then by Invariant 3 $u_j = \mathsf{up}(j) < j$ in $\mathrm{Up}(h_\sigma)$ and thus in the new tree $u_j = \mathsf{up}(j) < i_j$; in this case $i_j = \mathsf{in}(\mathsf{up}(i_j))$ in the new tree and hence i_j satisfies point 3 of Invariant 3. If $u_j \neq q$ then by Invariant 3 $u_j = \mathsf{up}(j) > j$ in $\mathrm{Up}(h_\sigma)$. As j and i_j are in the same subtree of u_j in $\mathrm{Up}(h_\sigma)$ Invariant 1 implies that $u_j > i_j$. It follows that in the new $\mathsf{dn}(i_j) < i_j < \mathsf{up}(i_j) = u_j$ and hence i_j satisfies point 2 of Invariant 3.

Node u_j: we only need to consider the case where $u_j \neq q$, as in the other case $\mathsf{up}(u_j)$ and $\mathsf{dn}(u_j)$ are not changed. Note that only $\mathsf{dn}(u_j)$ changes and that $\mathsf{up}(u_j)$ is the same in $\mathrm{Up}(h_\sigma)$ and the new tree. By Invariant 1 and the definition of h_λ we have $h_\theta(u_j) = h_\sigma(u_j) < h_\sigma(\mathsf{up}(u_j)) = h_\theta(\mathsf{up}(u_j))$. As shown for i_j we have $h_\theta(i_j) = h_\theta(\mathsf{dn}(u_j)) < h_\theta(u_j)$ in the new tree and thus $h_\theta(\mathsf{dn}(u_j)) < h_\theta(u_j) < h_\theta(\mathsf{up}(u_j))$. We now consider the two cases of Line 9:

$q < j < p$**:** as shown for node i_j it holds that $u_j < \mathsf{dn}(u_j) = i_j$ in the new tree. As j is on the mid-trail between p and q in $\mathrm{Up}(h_\sigma)$ so is u_j and thus $\mathsf{up}(u_j) < u_j$. Hence, $\mathsf{up}(u_j) < u_j < \mathsf{dn}(u_j)$ in the new tree.

$q < p < j$**:** recall that $u_j \neq q$. We have shown for node i_j that $\mathsf{dn}(u_j) = i_j < u_j$. As j is on the in-trail between p and q in $\mathrm{Up}(h_\sigma)$ so is u_j and thus either $q = \mathsf{up}(u_j) < u_j$ or $\mathsf{up}(u_j) > u_j$. In the first case u_j satisfies point 3 of Invariant 3 and point 2 of Invariant 3 in the new tree.

Node s^+: we only need to consider the case where $s^+ \neq q$, as in the other case $\mathsf{up}(s^+)$ and $\mathsf{dn}(s^+)$ are not changed. This implies that in the new tree $\mathsf{dn}(s^+) = j$. The details are similar to those for node j. We have seen there that $h_\theta(s^+) > h_\theta(j)$ and thus in the new tree $h_\theta(s^+) > h_\theta(\mathsf{dn}(s^+))$. As the $\mathsf{up}(s^+)$ is the same in the new tree as in $\mathrm{Up}(h_\sigma)$ it follows that $h_\theta(\mathsf{up}(s^+)) > h_\theta(s^+)$ also in the new tree. Now consider the two cases of Line 9:

$q < j < p$**:** as above this implies $j = \mathsf{dn}(s^+) < s^+$. If $\mathsf{up}(s^+) = q$ then $s^+ = \mathsf{in}(\mathsf{up}(s^+))$, $\mathsf{up}(s^+) < s^+$ and hence s^+ satisfies point 3 of Invariant 3. If $\mathsf{up}(s^+) \neq q$ then $s^+ < \mathsf{up}(s^+)$ by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ and hence s^+ satisfies point 2 of Invariant 3.

$q < p < j$**:** as above this implies $j = \mathsf{dn}(s^+) > s^+$. As s^+ is on the mid-trail from p to q in $\mathrm{Up}(h_\sigma)$ it follows that $s^+ > \mathsf{up}(s^+)$. Thus, s^+ satisfies point 2 of Invariant 3.

Node s^-: we only need to consider the case where $s^- \neq p$, as p has a minimum and does not affect Invariant 3. This implies that in the new tree $\mathsf{up}(s^-) = j$. The pointer $\mathsf{dn}(s^-)$ remains unchanged and thus $h_\theta(\mathsf{dn}(s^-)) < h_\theta(s^-)$. As $h_\theta(s^-) = h_\sigma(s^-) < h_\sigma(j) = h_\theta(j)$ we also have $h_\theta(s^-) < h_\theta(\mathsf{up}(s^-))$. Since $s^- = \mathsf{in}(j) = \mathsf{in}(\mathsf{up}(s^-))$ in the new tree it remains to show that $\mathsf{up}(s^-) = j$ and $\mathsf{dn}(s^-)$ are on the same side of s^- in both cases of Line 9.

$q < j < p$**:** j is on the mid-trail and s^- is on the in-trail between p and q, and thus by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ $j < s^-$. These conditions also imply that $\mathsf{dn}(s^-) < s^-$.

$q < p < j$**:** j is on the in-trail and s^- is on the mid-trail between p and q, and thus by Conditions $\mathsf{III}.1$ and $\mathsf{III}.2$ $s^- < j$. These conditions also imply that $s^- < \mathsf{dn}(s^-)$

In both cases s^- satisfies point 3 of Invariant 3.

All nodes for which the $\mathsf{up}(\cdot)$ or $\mathsf{dn}(\cdot)$ pointer changed satisfy the conditions of Invariant 3 in the new tree, and thus the new tree satisfies Invariant 3.

We now show that the new tree also satisfies Invariant 1. For any maximum, if the contents of the subtrees of that maximum are unchanged, then the condition of Invariant 1 continues to hold for this maximum. That is, for a maximum b, if the set of descendants of $\mathsf{dn}(b)$ is the same in $\mathrm{Up}(h_\sigma)$ and in the new tree, and if the set of nodes in the banana subtree rooted at b is the same in $\mathrm{Up}(h_\sigma)$ and in the new tree, and if $\mathrm{Bth}(b) < b$ (or $\mathrm{Bth}(b) > b$) in $\mathrm{Up}(h_\sigma)$ and in the new tree, then b satisfies the condition of Invariant 1 in the new tree, since it also satisfied it in $\mathrm{Up}(h_\sigma)$. This is the case for most of the nodes:

- $\mathrm{Bth}(q) = p$ in $\mathrm{Up}(h_\sigma)$ and $\mathrm{Bth}(q) = i$ in the new tree, but both $q < p$ and $q < i$. The set of nodes in either subtree of q remains unchanged.
- For any node on the trail from u_j to q (except u_j and q) and any node on the trail from s^+ to q (excluding s^+ and q) the set of nodes in either subtree remains unchanged, and so does the $\mathrm{Bth}(\cdot)$ of these nodes.
- The same holds for any descendant of j in $\mathrm{Up}(h_\sigma)$, for s^- and any descendant of s^-.
- For any node not in the banana subtree of q the contents of the subtrees and $\mathrm{Bth}(\cdot)$ is unchanged.

Thus, we only need to show that u_j, j and s^+ do not violate the condition of Invariant 1. Furthermore, we only need to consider the case where $u_j \neq q$ and $s^+ \neq q$.

Node u_j: the banana subtree of u_j is the same in $\mathrm{Up}(h_\sigma)$ and in the new tree, as is $\mathrm{Bth}(u_j)$. The node $\mathsf{dn}(u_j) = i_j$ in the new tree is a descendant of j in $\mathrm{Up}(h_\sigma)$ and $j = \mathsf{dn}(u_j)$ in $\mathrm{Up}(h_\sigma)$. The descendants of i_j are descendants of $\mathsf{dn}(u_j)$ in both $\mathrm{Up}(h_\sigma)$ and the new tree. As u_j does not gain any descendants, it follows that u_j satisfies the condition of Invariant 1 in the new tree.

Node s^+: the banana subtree of s^+ is the same in $\mathrm{Up}(h_\sigma)$ and in the new tree, as is $\mathrm{Bth}(s^+)$. Through the change of $\mathsf{dn}(s^+)$ to j, $\mathsf{dn}(s^+)$ gains new descendants in the new tree. These are the nodes on the trail between p and j, along with their banana subtrees, and the nodes on the trail from i to j. By Invariant 1 and Conditions III.1 and III.2 for $\mathrm{Up}(h_\sigma)$ these are all on the same side of s^+ as $\mathsf{dn}(s^+)$ in $\mathrm{Up}(h_\sigma)$. It follows that s^+ satisfies the condition of Invariant 1 in the new tree.

Node j: The two cases $q < j < p$ and $q < p < j$ are symmetric. We give the proof for the first case and assume $q < j < p$. The pointers $\mathsf{mid}(j)$ and $\mathsf{dn}(j)$ are swapped going from $\mathrm{Up}(h_\sigma)$ to the new tree (see the assignment in Line 8). The banana subtree rooted at j in the new tree consists of descendants of s^+ and the descendants of $\mathsf{dn}(j)$ in $\mathrm{Up}(h_\sigma)$. By Invariant 1 and Conditions III.1 and III.2 for $\mathrm{Up}(h_\sigma)$, each node v in this set satisfies $j < v$. Note that $\mathrm{Bth}(j) = p$ in the new tree and thus $j < \mathrm{Bth}(j)$ by assumption. The descendants of $\mathsf{dn}(j)$ in the new tree are descendants of nodes on the mid-trail of j in $\mathrm{Up}(h_\sigma)$. Each node t in this set satisfies $t < j$ by Invariant 1 for $\mathrm{Up}(h_\sigma)$. Thus, in the new tree j satisfies the condition of Invariant 1.

We have shown that all nodes satisfy the condition of Invariant 1 and thus the new up-tree satisfies Invariant 1.

It remains to show that the new up-tree also satisfies Invariant 2. Invariant 2 can be violated only by minima for which $\mathsf{low}(\mathsf{dth}(\cdot))$ changes. This changes for i, p and any node u with $\mathsf{low}(\mathsf{dth}(\cdot)) = p$ in $\mathrm{Up}(h_\sigma)$.

- In the new tree $\mathsf{low}(\mathsf{dth}(p)) = i$, and $h_\theta(p) > h_\theta(i)$ by definition of h_θ.
- As the algorithm sets $\mathsf{dth}(i) = q$ (see Line 22) in the new tree $\mathsf{low}(\mathsf{dth}(i)) = \mathsf{low}(q)$. Since either $h_\sigma(i) = h_\theta(i)$ or $h_\sigma(p) = h_\theta(p)$ and since i and p are contiguous in both h_σ and h_θ, $h_\sigma(p) > h_\sigma(\mathsf{low}(q))$ implies that $h_\theta(i) > h_\sigma(\mathsf{low}(q)) = h_\theta(\mathsf{low}(q)) = h_\theta(\mathsf{low}(\mathsf{dth}(i)))$.
- The nodes u for which we set $\mathsf{low}(\mathsf{dth}(u)) = i$ have $\mathsf{low}(\mathsf{dth}(u)) = p$ in $\mathrm{Up}(h_\sigma)$. They satisfy $h_\sigma(u) = h_\theta(u) > h_\sigma(\mathsf{low}(\mathsf{dth}(p))$, and contiguity of i and p in h_σ they satisfy $h_\theta(u) > h_\sigma(i)$. As $h_\theta(i) \le h_\sigma(i)$, it follows that $h_\theta(u) > h_\theta(i) = h_\theta(\mathsf{low}(\mathsf{dth}(u))$.

All minima satisfy the condition of Invariant 2 and thus the new tree satisfies Invariant 2. We have shown that the new tree satisfies Invariants 1 to 3 for h_θ, which concludes the proof that it is the unique up-tree for h_θ. □

A.6 Correctness of Scenarios A and B

So far, we have described the local operations in terms of small changes in function value, where the involved items are contiguous in function value before and after the operation. However, Scenarios A and B described in section 5.2 also need to deal with larger changes in value, involving not necessarily contiguous items. We start by describing the conditions under which a change in the value of a maximum (or a minimum by lemma 5.2) does not affect the corresponding banana tree. This is stated formally in the following lemma.

Lemma A.17 (Unaffected Banana Tree)**.** *Let j be a maximum. Let h_σ and h_θ be maps that differ only in the value of j such that all other items have the same criticality. If it holds that* $\max\{h_\sigma(\mathsf{in}(j)), h_\sigma(\mathsf{mid}(j)), h_\sigma(\mathsf{dn}(j))\} < h_\theta(j) < h_\sigma(\mathsf{up}(j))$ *then* $\mathrm{Up}(h_\sigma)$ *and* $\mathrm{Up}(h_\theta)$ *are identical.*

Proof. By Invariants 1 to 3, we know that $\max\{h_\sigma(\mathsf{in}(j)), h_\sigma(\mathsf{mid}(j)), h_\sigma(\mathsf{dn}(j))\} < h_\sigma(j) < h_\sigma(\mathsf{up}(j))$. The up-tree $\mathrm{Up}(h_\sigma)$ satisfies Invariant 1 for the map h_θ, since the order of items along the interval is independent of h_σ and h_θ. It also satisfies Invariant 2 for h_θ, as $h_\sigma(u) = h_\theta(u)$ for all minima u in h_σ and h_θ, and the set of minima is the same in h_σ and h_θ. Finally, it also satisfies Invariant 3, which follows from $\max\{h_\sigma(\mathsf{in}(j)), h_\sigma(\mathsf{mid}(j)), h_\sigma(\mathsf{dn}(j))\} < h_\theta(j) < h_\sigma(\mathsf{up}(j))$ and the fact that the order of items along the interval is independent of h_σ and h_θ. By corollary A.8 there is a unique banana tree for h_θ that satisfies Invariants 1 to 3, and thus $\mathrm{Up}(h_\sigma) = \mathrm{Up}(h_\theta)$. □

We now show that the algorithms given for Scenarios A and B indeed transform $\mathrm{Up}(f)$ into $\mathrm{Up}(g)$, where the maps f and g differ in the value of item j. Recall that in Scenario A item j is non-critical in f and $f(j) < g(j)$, and that in Scenario B item j is a maximum in f and decreases its value until it is non-critical in g. For Scenarios A and B we assume that j is not an endpoint, i.e., $j \in [2, m-1]$ and we give an additional algorithm for updating the value of an endpoint below.

We restate the algorithm for Scenario A, where we now also account for updates to items that lead to an endpoint changing from down-type to up-type (see Line 2).

```
1  if $g(j) > f(j+1)$ then
2    if $j+2$ is a hook then cancel or slide endpoint $j+1$ to $j$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$
3    else if $f(j+1) < f(j+2)$ then anti-cancel $j$ and $j+1$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$
4                                   else slide $j+1$ to $j$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$
5    endif;
6    set $q = \mathsf{up}(j)$ in $\mathrm{Up}(f)$;
7    while $f(q) < g(j)$ do interchange $j$ and $q$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$;
8                            $q = \mathsf{up}(j)$ in $\mathrm{Up}(f)$
9    endwhile;
```

```
endif.
```

The following lemma states that this algorithm correctly updates Up(f) to obtain Up(g).

Lemma A.18 (Correctness of Scenario A)**.** *Let f and g be maps with $f(j) < g(j)$ and $f(i) = g(i)$ for all $i \neq j$, and $f(j-1) < f(j) < f(j+1)$. Applying the algorithm for Scenario A to* Up(f) *yields* Up(g)*.*

Proof. We show how the up-tree Up(f) is modified throughout the algorithm following the homotopy h_λ until the up-tree becomes Up(g). If $g(j) < f(j+1)$ then item j is non-critical in f and g, Up(f) = Up(g), and the algorithm does not change the up-tree. Assume $g(j) > f(j+1)$ for the remainder of the proof. By this assumption there exists an h_{θ_0} with $0 < \theta_0 \leq 1$ such that j and $j+1$ are contiguous in h_{θ_0}. If $j+2$ is a hook, then $j+1$ is down-type in f and up-type in h_{θ_0} and by lemma A.12 applying `cancel-or-slide-endpoint`$(j, j+1)$ yields Up(h_{θ_0}). If $j+2$ is not a hook and $f(j+1) < f(j+2)$, then j and $j+1$ are non-critical in f and a maximum and minimum, respectively, in h_{θ_0}. By lemma A.13 executing an anti-cancellation between j and $j+1$ yields the up-tree Up(h_{θ_0}). If $j+2$ is not a hook and $f(j+1) > f(j+2)$, then $j+1$ is a maximum in f, non-critical in h_{θ_0} and j is a maximum in h_{θ_0}. By lemma A.9 executing a slide from $j+1$ to j yields the up-tree Up(h_{θ_0}).

Let q_i be $\mathsf{up}(j)$ at the beginning of the i-th iteration of the loop in Line 7. We show by induction that after every iteration i of the loop either (1) the current up-tree corresponds to a map $h_{\theta_{i+1}}$ where q_i and j are contiguous and $h_{\theta_{i+1}}(q_i) < h_{\theta_{i+1}}(j)$ or (2) the current up-tree is Up(g). For the base case $i = 1$, if $f(q_1) > g(j)$, then the up-tree Up(h_{θ_0}) = Up(g) by lemma A.17. Otherwise, there exists a $\theta_1 \in (\theta_0, 1]$ such that $h_{\theta_1}(q_1) < h_{\theta_1}(j)$ and q_1 and j are contiguous in h_{θ_1}. By lemma A.17 and lemma A.15 interchanging q_1 and j yields Up(h_{θ_1}). Now assume that at the beginning of some iteration i the current up-tree is Up($h_{\theta_{i-1}}$), with $h_{\theta_{i-1}}(j) \in [\max\{f(\mathsf{in}(j)), f(\mathsf{mid}(j)), f(\mathsf{dn}(j))\}, f(q_i)]$. If $f(q_i) > g(j)$, then the up-tree Up(h_{θ_i}) = Up(g) by lemma A.17. Else, there exists a $\theta_{i+1} \in (\theta_i, 1]$ such that $h_{\theta_{i+1}}(q_i) < h_{\theta_{i+1}}(j)$ and q_i and j are contiguous in $h_{\theta_{i+1}}$. Again, interchanging q_i and j yields the up-tree Up($h_{\theta_{i+1}}$).

There exists an iteration I where $f(q_I) > g(j)$, at the latest when q_I is the special root, and the loop terminates. Then, by lemma A.17 the Up($h_{\theta_{I-1}}$) = Up(g). □

The following algorithm updates the banana trees in Scenario B, now including updates that change up-type items to down-type items (see Line 5).

```
1  loop $q = \arg\max\{f(\mathsf{dn}(j)), f(\mathsf{in}(j)), f(\mathsf{mid}(j))\}$ in Up($f$);
2    if $f(q) > f(j+1)$ then interchange $q$ and $j$ in Up($f$) and Dn($f$)
3                         else exit endif
4  forever;
5  if $j+2$ is a hook then cancel or slide endpoint $j+1$ to $j$ in Up($f$) and Dn($f$)
6  else if $f(j+1) < f(j+2)$ then cancel $j$ with $j+1$ in Up($f$) and Dn($f$)
7                                else slide $j$ to $j+1$ in Up($f$) and Dn($f$)
8  endif.
```

The next lemma states that this algorithm correctly updates Up(f) to obtain Up(g).

Lemma A.19 (Correctness of Scenario B)**.** *Let f and g be maps with $f(j) > g(j)$ and $f(i) = g(i)$ for all $i \neq j$, and $f(j-1) < f(j) > f(j+1)$. Applying the algorithm for Scenario B to* Up(f) *yields* Up(g)*.*

Proof. The proof is analogous to that of lemma A.18: after each iteration i of the loop, there is a θ_i such that q_i and j are contiguous in h_{θ_i} and the current up-tree is $\mathrm{Up}(h_{\theta_i})$. Let $\mathrm{Up}(h_\gamma)$ be the up-tree after the loop, where in h_γ the items j and q are contiguous in value. If $j+2$ is a hook, then $j+1$ is an up-type endpoint in h_γ and a down-type endpoint in g, since $g(j) < g(j+1)$. Then, by lemma A.12 and the fact that non-critical items are not represented in the up-tree applying `cancel-or-slide-endpoint`$(j, j+1)$ to $\mathrm{Up}(h_\gamma)$ yields $\mathrm{Up}(g)$. Similarly, if $j+2$ is not a hook and $f(j+1) < f(j+2)$, then j and $j+1$ are non-critical in g as $f(j-1) < g(j) < f(j+1) < f(j+2)$, and canceling j with $j+1$ yields $\mathrm{Up}(g)$ by lemma A.11. Finally, if $j+2$ is not a hook and $f(j+1) > f(j+2)$, then $j+2$ was non-critical in f and is a maximum in g, and sliding j to $j+1$ yields $\mathrm{Up}(g)$ by lemma A.9. □

To conclude, we state an algorithm for updating the banana trees under the change of value of the endpoint m. The algorithm for the other endpoint of the interval is symmetric. We assume that m is up-type in f and down-type in g, i.e., $f(m) < f(m-1) < g(m)$. There is also the reverse case, in which m is down-type in f and up-type in g.

```
1  Set $q = \mathtt{dn}(m)$ in $\mathrm{Dn}(f)$;
2  while $q$ is a maximum in $-f$ and $f(q) < g(m)$ do
     interchange $m$ and $q$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$;
3    $q = \mathtt{dn}(m)$ in $\mathrm{Dn}(f)$
4  endwhile;
5  if $f(m-1) < g(m)$ then cancel or slide endpoint $m$ to $m-1$ endif;
6  Set $q = \mathtt{up}(m)$ in $\mathrm{Up}(f)$;
7  while $f(q) < g(m)$ do interchange $m$ and $q$ in $\mathrm{Up}(f)$ and $\mathrm{Dn}(f)$;
8                         $q = \mathtt{up}(m)$ in $\mathrm{Up}(f)$
9  endwhile.
```

In the first loop we exploit the coupling of interchanges between the up-tree and down-tree: we find the maximum with which to interchange m in $\mathrm{Dn}(f)$, and perform the corresponding interchange of minima in $\mathrm{Up}(f)$. The purpose of this loop is to prepare the banana tree for switching m from an up-type to a down-type item, as described in appendix A.3. The value of the hook $m+1$ is defined to have value $f(m+1) = m + \varepsilon$ and $g(m+1) = g(m) - \varepsilon$, i.e., its value is adjusted alongside that of m, which justifies interchanging m only with $\mathtt{dn}(m)$ in the first loop, rather than with $\mathtt{in}(m)$, $\mathtt{mid}(m)$ and $\mathtt{dn}(m)$ as we have done in Scenario B. For completeness we state in the next lemma that this algorithm correctly maintains the up-tree; we omit the proof, which is analogous to that of lemmas A.18 and A.19.

Lemma A.20 (Local Change at an Endpoint)**.** *Let f and g be maps differing only in the value of the endpoint m, such that $f(m) < f(m-1) < g(m)$. Applying the previous algorithm to* $\mathrm{Up}(f)$ *yields* $\mathrm{Up}(g)$*.*

B Correctness of Topological Maintenance

B.1 Splitting Banana Tree

In this section we prove that the algorithm Split described in section 5.3 correctly splits a banana tree. The proof will be by induction over the iterations of the loop in Split, with the base case in lemma B.1 and the induction step in lemma B.2. We first prove these two lemmas and then state the result in theorem B.3.

Given a list of items, its associated map f and an item ℓ, the algorithm `Split` computes from $\mathrm{Up}(f)$ two new up-trees $\mathrm{Up}(g)$ and $\mathrm{Up}(h)$, where the former contains the items up to ℓ and the latter the remaining items. We define x to be the midpoint of ℓ and $\ell+1$ with $f(x) = \frac{1}{2}(f(\ell) + f(\ell+1))$. We write Stack_i, p_i, q_i for Stack, p, q returned by TopBanana in the i-th iteration of Split. The up-trees $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$ denote the left and right up-trees after the i-th iteration, with initial up-trees $\mathrm{Up}(g_0)$, $\mathrm{Up}(h_0)$. One of the initial up-trees consists of the new special root and the dummy leaf α. We assume that this is the tree $\mathrm{Up}(h_0)$ with special root β_h and that $\mathrm{Up}(g_0) = \mathrm{Up}(f)$. This is equivalent to assuming that the top banana on the stacks is on the right spine of $\mathrm{Up}(f)$, where the top banana is the banana closest to the root such that x lies in some panel of the corresponding window. The other case where the top banana is on the left spine of $\mathrm{Up}(f)$ is symmetric. We further define α such that it is the rightmost item of $\mathrm{Up}(g_i)$ if it is in $\mathrm{Up}(g_i)$ and the leftmost item of $\mathrm{Up}(h_i)$ if it is in $\mathrm{Up}(h_i)$. Finally, let I be the total number of bananas on the stack, i.e., the number of iterations of Split.

Lemma B.1 (Base Case). *At the beginning of the first iteration it holds that*

1. *for all nodes u in $\mathrm{Up}(g_0)$ and all nodes v in $\mathrm{Up}(h_0)$ we have $u < v$,*
2. *$\mathrm{Up}(g_0)$ and $\mathrm{Up}(h_0)$ satisfy Invariants 1 to 3,*
3. *any node u that is not on any stack and does not have an ancestor on any stack is in $\mathrm{Up}(g_0)$ if $u < x$ and in $\mathrm{Up}(h_0)$ if $u > x$,*
4. *$f(\alpha) < f(x)$, $f(q_j) < f(\mathsf{in}(\alpha))$ and $f(q_j) < f(\mathsf{mid}(\alpha))$ at the beginning of the iteration for all $j \in [1, I]$ with $(p_j, q_j) \in L_{\mathrm{up}} \cup R_{\mathrm{up}} \cup M_{\mathrm{up}}$,*
5. *all bananas $(p_j, q_j) \in L_{\mathrm{up}} \cup R_{\mathrm{up}} \cup M_{\mathrm{up}}$ for $j \in [1, I]$ are in $\mathrm{Up}(g_0)$ if α is in $\mathrm{Up}(h_0)$ and in $\mathrm{Up}(h_1)$ if α is in $\mathrm{Up}(g_1)$,*
6. *if $\mathit{Stack}_1 \in \{L_{\mathrm{dn}}, R_{\mathrm{dn}}\}$ then q_1 is in the same tree as α,*
7. *the in-trail between α and $\mathsf{dth}(\alpha)$ is empty and α is on the spine.*

Proof. Assume that $\mathrm{Up}(g_0) = \mathrm{Up}(f)$. Claim 1 holds by definition of α and the special root of $\mathrm{Up}(h_0)$. The up-tree $\mathrm{Up}(f) = \mathrm{Up}(g_0)$ is the input to the algorithm and satisfies Invariants 1 to 3 by assumption. The topmost banana (p_1, q_1) has q_1 on the right spine of $\mathrm{Up}(g_0)$ and $q_1 < x$, which implies that nodes u that are not descendants of q_1 satisfy $u < q_1 < x$. These are precisely the nodes in $\mathrm{Up}(g_0)$ that have no ancestor on any stack. The nodes in $\mathrm{Up}(h_0)$ also have no ancestor on any stack, but $\alpha > x$ by definition, and so is the special root of $\mathrm{Up}(h_0)$. This implies claim 3. The inequality $f(\alpha) < f(x)$ follows from $f(\alpha) < f(p_1) < f(x)$, where the last inequality holds since x is in the in-panel or mid-panel of $W(p_1, q_1)$. The pointers $\mathsf{in}(\alpha)$ and $\mathsf{mid}(\alpha)$ both point to the special root of $\mathrm{Up}(h_0)$, which by definition has greater value than all q_j for $j \in [1, I]$ with $q_j \in L_{\mathrm{up}} \cup R_{\mathrm{up}} \cup M_{\mathrm{up}}$. Thus, all three inequalities of claim 4 hold. To see that claim 5 holds simply note that all nodes on the stacks are in $\mathrm{Up}(g_0)$ and α is in $\mathrm{Up}(h_0)$. Claim 6 holds trivially, as the topmost banana (p_1, q_1) cannot be on L_{dn} or R_{dn}: this would put x in the in-panel or mid-panel of the banana that (p_1, q_1) is nested in, which contradict (p_1, q_1) being the topmost banana. Finally, α is the only other node in $\mathrm{Up}(h_0)$ besides the special root, and thus both trails between α and $\mathsf{dth}(\alpha)$ are empty and α is on the spine. □

Lemma B.2 (Inductive Step). *If at the beginning of any iteration i of* Split *it holds that*

1. *for all nodes u in $\mathrm{Up}(g_{i-1})$ and all nodes v in $\mathrm{Up}(h_{i-1})$ we have $u < v$,*

2. *$\mathrm{Up}(g_{i-1})$ and $\mathrm{Up}(h_{i-1})$ satisfy Invariants 1 to 3,*
3. *any node u that is not on any stack and does not have an ancestor on any stack is in g_{i-1} if $u < x$ and in h_{i-1} if $u > x$,*
4. *$f(\alpha) < f(x)$, $f(q_j) < f(\mathsf{in}(\alpha)$ and $f(q_j) < f(\mathsf{mid}(\alpha))$ at the beginning of the iteration for all $j \in [i, I]$ with $(p_j, q_j) \in L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$,*
5. *all bananas $(p_j, q_j) \in L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$ for $j \in [i, I]$ are in $\mathrm{Up}(g_i)$ if α is in $\mathrm{Up}(h_i)$ and in $\mathrm{Up}(h_i)$ if α is in $\mathrm{Up}(g_i)$,*
6. *if $Stack_i \in \{L_{\text{dn}}, R_{\text{dn}}\}$ then q_i is in the same tree as α,*
7. *the in-trail between α and $\mathsf{dth}(\alpha)$ is empty and α is on the spine,*

then after the i-th iteration

1. *for all nodes u in $\mathrm{Up}(g_i)$ and all nodes v in $\mathrm{Up}(h_i)$ we have $u < v$,*
2. *$\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$ satisfy Invariants 1 to 3,*
3. *any node u that is not on any stack and does not have an ancestor on any stack is in g_i if $u < x$ and in h_i if $u > x$,*
4. *$f(\alpha) < f(x)$, $f(q_j) < f(\mathsf{in}(\alpha))$ and $f(q_j) < f(\mathsf{mid}(\alpha))$ for all $j \in [i+1, I]$ with $(p_j, q_j) \in L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$,*
5. *all bananas $(p_j, q_j) \in L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$ for $j \in [i+1, I]$ are in $\mathrm{Up}(g_i)$ if α is in $\mathrm{Up}(h_i)$ and in $\mathrm{Up}(h_i)$ if α is in $\mathrm{Up}(g_i)$,*
6. *if* TopBanana *returns $Stack_{i+1} \in \{L_{\text{dn}}, R_{\text{dn}}\}$ then q_{i+1} is in the same tree as α,*
7. *the in-trail between α and $\mathsf{dth}(\alpha)$ is empty and α is on the spine.*

Proof. There are in total six cases:

1. $Stack = L_{\text{up}}$,
2. $Stack = R_{\text{up}}$,
3. $Stack = M_{\text{up}}$ and $q < x < p$,
4. $Stack = M_{\text{up}}$ and $p < x < q$,
5. $Stack = L_{\text{dn}}$,
6. $Stack = R_{\text{dn}}$.

The first two cases are symmetric, as are the two cases with $Stack = M_{\text{up}}$ and the last two cases. We prove the claim for cases 1, 3 and 5. In the following we assume that the seven conditions hold.

$Stack_i = L_{\text{up}}$**:** SPLIT executes DOINJURY. By definition of L_{up} $q_i < p_i < x$. The node q_i is on a spine, as otherwise the banana it is nested in would contain x in its in- or mid-panel. This is not possible, since this banana would then be on the stacks above (p_i, q_i). Furthermore, $q_i < p_i$ implies that q_i is on a right spine and it is thus in the left tree. The fourth claim follows immediately from the fourth assumption, as the value of α remains unchanged and L_{up}, R_{up}, M_{up} can be merged into a sorted stack by lemma 5.4.

Nodes j with $j > x$ on the in-trail between p_i and q_i are removed and inserted between α and $\mathsf{mid}(\alpha)$. These nodes and the nodes in their banana subtrees are the rightmost nodes of $\text{Up}(g_i)$, which implies claim 1. Since we do not modify the in-trail between α and $\mathsf{dth}(\alpha)$ and it is assumed to be empty at the beginning of the iteration, this trail is also empty after the iteration, which proves claim 7.

Denote by j^- the lowest and highest among the moved nodes in terms of function value, respectively. Conditions III.1 and III.2 hold in $\text{Up}(g_{i-1})$ and $\text{Up}(h_{i-1})$, and thus the nodes j form a contiguous section of the in-trail with $j^+ = \mathsf{in}(q_i)$. Write a for the highest node on the in-trail between p_i and q_i that is not moved. Both the in-trail originally containing the nodes j and the mid-trail between α and $\mathsf{dth}(\alpha)$ are right trails. By the assumption that nodes in $\text{Up}(g_i)$ are less than nodes in $\text{Up}(h_i)$ it follows that $\alpha = \mathsf{dn}(j^-) < j^- < \cdots < j^+ < \mathsf{up}(j^+)$. The item x is in the in-panel of the window $W(p_i, q_i)$ corresponding to the banana (p_i, q_i) in $\text{Up}(g_{i-1})$, and by Conditions III.1, III.2 and Invariant 3 $x < j^-$ also implies $f(x) < f(j^-)$. This, together with the fourth assumption implies that $f(\alpha) = f(\mathsf{dn}(j^-)) < f(x) < f(j^-) < \cdots < f(j^+) < f(\mathsf{up}(j^+))$. Thus, Invariant 3 holds in the new trees.

Since nodes pointed to by $\mathsf{low}(\mathsf{dth}(\cdot))$ only change for $\text{Bth}(j)$ where j is one of the moved nodes, namely from p_i to α, Invariant 2 holds in the new trees by the assumption that $f(\alpha) < f(p_i)$.

Invariant 1 continues to hold in the tree that originally contained the nodes j, since nodes are only removed from subtrees. The nodes j themselves only gain α as a descendant of $\mathsf{dn}(j)$ or as $\mathsf{dn}(j)$ itself. In addition, $\alpha < j$ by definition of α and $j < \text{Bth}(j)$ by definition of right trail. Thus, the nodes j also satisfy the condition of Invariant 1. The nodes j are inserted as descendants of $\mathsf{dn}(t)$, where $t = \mathsf{mid}(\alpha)$ before the insertion. Since $j < \mathsf{dn}(t)$ by assumption item 1, the condition of Invariant 1 continues to holds for t, and also holds for the remaining nodes of this tree. It follows that both trees satisfy Invariant 1 after the i-th iteration.

The nodes u in $\text{Up}(h_{i-1})$ that are not on any stack and do not have an ancestor on any stack satisfy $u > x$. The nodes j and the nodes in their banana subtrees are also greater than x. Thus, $\text{Up}(h_{i-1})$ contains no node u that is not on any stack and has no ancestor on any stack with $u < x$. In the left tree $\text{Up}(g_i)$ the node a and the nodes in its subtree are the rightmost nodes of a, since a is on the right spine. Other nodes in $\text{Up}(g_i)$ are thus less than x. Note that $a < x$ by definition of a. The node a is either a minimum, a maximum and not on any stack, or a maximum and on some stack. In the first two cases all nodes in $\text{Up}(g_i)$ are less than x. In the third case, all nodes except possibly descendants of a are less than x, and descendants of a have an ancestor on the stack. Thus, claim 3 holds.

To see that claim 5 holds, recall that the bananas in L_{up}, R_{up} and M_{up} are nested into each other. Call the topmost banana on these stacks (p_ℓ, q_ℓ). It must be nested into (p_i, q_i) and is thus on the in-trail between p_i and q_i in $\text{Up}(g_i)$, as otherwise x would be in the mid-panel of $W(p_i, q_i)$, which contradicts $Stack_i = L_{\text{up}}$. This implies $q_i < p_i$ and $q_i < x$. It follows that q_i remains in the left tree, while α remains in the right tree. This proves claim 5.

We now prove claim 6. Assume that $\mathit{Stack}_{i+1} = L_{\text{dn}}$. Then, by definition $p_{i+1} < q_{i+1}$. Thus, q_{i+1} cannot be on the in-trail between p_i and q_i. It can also not be on any other trail, since then $q_{i+1} < p_i < x$, which implies that x is not in the out-panel of $W(p_{i+1}, q_{i+1})$. It follows that p_{i+1} and q_{i+1} cannot be in the left tree. Now assume that $\mathit{Stack}_{i+1} = R_{\text{dn}}$. Then, by definition $q_{i+1} < p_{i+1}$. If q_{i+1} is in $\text{Up}(g_i)$, then $q_{i+1} < x$, as all maxima greater than x have been moved to $\text{Up}(h_i)$. This contradicts $W(p_{i+1}, q_{i+1})$ having x in its out-panel, which implies that q_{i+1} must be in the right tree. In both cases, q_{i+1} is in the right tree, which is the same tree as α.

$\mathit{Stack}_i = M_{\text{up}}$ **and** $q_i < x < p_i$**:** SPLIT executes DOFATALITY. By definition of M_{up} $q_i < x < p_i$. This case is similar to the case $\mathit{Stack}_i = L_{\text{up}}$ and we mainly point out the differences. As above, q_i is on the left spine of $\text{Up}(g_{i-1})$. Since x is in a panel of the window (p_i, q_i) it holds that $f(p_i) < f(x)$. The fourth claim then follows immediately from the third assumption, as $f(\alpha) < f(p_i)$ after the iteration and L_{up}, R_{up}, M_{up} can be merged into a sorted stack by lemma 5.4.

All nodes internal to the in-trail between p_i and q_i are moved to the right tree, as are the nodes j with $j > x$ on the mid-trail between p_i and q_i. The node α replaces p_i as $\text{Bth}(q_i)$. As q_i is on a spine the nodes that are moved to the right tree are the rightmost nodes of $\text{Up}(g_{i-1})$, and together with assumption 1 this implies claim 1, i.e., that the nodes in $\text{Up}(h_i)$ are all greater than the nodes in $\text{Up}(g_i)$. The in-trail between α and $\mathsf{dth}(\alpha)$ in $\text{Up}(g_i)$ is the in-trail between α and q_i, which is empty as all nodes on the in-trail beginning at q_i have been moved to $\text{Up}(h_i)$.

Invariants 1 and 3 hold in $\text{Up}(g_i)$ as $f(\alpha) < f(p_i) < f(q_i)$ by assumption 4 and because the order of maxima and nodes in their subtrees is unchanged. By the assumption that $f(q_i) < f(\mathsf{in}(\alpha)))$ and $f(q_i) < f(\mathsf{mid}(\alpha))$, where $\mathsf{in}(\alpha)$ and $\mathsf{mid}(\alpha)$ are the pointers in $\text{Up}(h_{i-1})$, the nodes j that are moved to the right tree satisfy $f(j) < f(\mathsf{up}(j))$. Those nodes j that are on a left trail in $\text{Up}(g_{i-1})$ are on a left trail in $\text{Up}(h_i)$, and those on a right trail are also on a right trail in $\text{Up}(h_i)$. It follows that $\text{Up}(h_i)$ also satisfies Invariants 1 and 3. In $\text{Up}(g_i)$, only for nodes with $\mathsf{low}(\mathsf{dth}(\cdot)) = p_i$ does this pointer change, namely to α, and $f(\alpha)$ is set to $f(p_i) - \varepsilon$. These nodes satisfy the condition of Invariant 2, as does α itself. In $\text{Up}(h_i)$ the node p_i satisfies the condition of Invariant 2, as $f(p_i) > f(\alpha)$. The nodes that are moved to the right tree also satisfy this condition, since $\mathsf{low}(\mathsf{dth}(\cdot))$ remains unchanged. Other nodes u with $\mathsf{low}(\mathsf{dth}(u)) = p_i$ in $\text{Up}(g_i)$ were already on the mid-trail between α and $\mathsf{dth}(\alpha)$ in $\text{Up}(h_{i-1})$. If $f(\mathsf{low}(\mathsf{dth}(u))) < f(p_i)$ then the windows $W(u, \mathsf{dth}(u))$ would have x in their out-panel, which places them in L_{dn} or R_{dn}. But by assumption 4 $f(\mathsf{dth}(u)) > f(q_i)$ which implies that they would have been processed in an earlier iteration. This is a contradiction and thus $f(\mathsf{low}(\mathsf{dth}(u))) > f(p_i)$, i.e., they satisfy the condition of Invariant 2. For no other node of $\text{Up}(h_i)$ does the node $\mathsf{low}(\mathsf{dth}(\cdot))$ change from what it was in $\text{Up}(h_{i-1})$, so Invariant 2 holds in $\text{Up}(h_i)$. This proves claim item 2.

For claim 3 observe that nodes u in $\text{Up}(g_i)$ satisfy $u < x$, as any nodes with $u > x$ are moved to h_i. Nodes v with $v < x$ in $\text{Up}(h_i)$ have as ancestor the highest node that is moved from the mid-trail; if such nodes v exist, then this highest node itself must be in either R_{up} or M_{up}, and thus the only nodes in $\text{Up}(h_i)$ with $v < x$ have an ancestor on some stack. This highest node can also be the only next node in $L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$, so claim 5 follows.

To see that claim 6 holds note that the next banana with x in the out-panel of the corresponding window can only be on the mid-trail between α and q_i in $\text{Up}(g_i)$, similar to the case $\mathit{Stack}_i = L_{\text{up}}$.

$Stack_i = L_{\text{dn}}$**:** Split executes DoScare, which sets $f(\alpha) = f(p_i) + \varepsilon$ and then executes an interchange of minima between α and p_i. The nodes q_i and α are in the same tree by assumption, and they must be in $\text{Up}(g_{i-1})$, as x is in the out-panel of $W(p_i, q_i)$ and the in-trail between α and $\mathsf{dth}(\alpha)$ is assumed to be empty. Going from $\text{Up}(g_{i-1})$ to $\text{Up}(g_i)$ and from $\text{Up}(h_{i-1})$ to $\text{Up}(h_i)$ does not change the set of nodes contained in each tree. In fact, $\text{Up}(h_{i-1}) = \text{Up}(h_i)$. Claims 1 and 5 follow immediately from the respective assumption. The set of nodes internal to the in-trail between α and $\mathsf{dth}(\alpha)$ after the interchange of minima is a subset of the nodes internal to the in-trail between α and $\mathsf{dth}(\alpha)$ before the iteration. The latter is empty by assumption 7. Furthermore, α is on the spine before iteration i by 7 and it remains on the spine through the interchange, so the claim 7 holds.

We now show claim 2. Note that q_i must be on the mid-trail between $\mathsf{dth}(\alpha)$ and α, i.e., $\mathsf{low}(q_i) = \alpha$, as otherwise x would not be in the out-panel of $W(p_i, q_i)$. We first show that there is no other minimum with value between $f(\alpha)$ and $f(p_i)$ whose $\mathsf{dth}(\cdot)$ pointer points to a maximum on the banana spanned by α and $\mathsf{dth}(\alpha)$. The in-trail between α and $\mathsf{dth}(\alpha)$ is empty by assumption. Let s be a minimum with $f(\alpha) < f(s) < f(p_i)$ and with $\mathsf{dth}(s)$ on the mid-trail between α and $\mathsf{dth}(\alpha)$. The banana $(s, \mathsf{dth}(s))$ has x in its out-panel and $s < \mathsf{dth}(s)$, and thus must be on the stack L_{dn}. It must be on L_{dn} below (p_i, q_i), as otherwise it would have been processed before and $f(\alpha) > f(s)$ at the beginning of the iteration. Since L_{dn} is sorted by lemma 5.4 this implies $f(s) > f(p_i)$, which is a contradiction. It follows that there is no other minimum that could be interchanged with α in place of p_i. Setting the value of α to $f(p_i) + \varepsilon$ leads to a violation of Invariant 2, but this is fixed by the interchange of minima. Thus, $\text{Up}(g_i)$ and $\text{Up}(h_i)$ satisfy Invariants 1 to 3.

To prove claim 3 we consider the nodes which no longer have an ancestor on any stack or are no longer on any stack themselves. The nodes no longer on any stack are p_i and q_i, which were taken of L_{dn}. These satisfy $p_i < x$ and $q_i < x$ by definition of L_{dn}. The remaining nodes to consider are nodes with ancestor q_i. Descendants u of q_i that were in the banana subtree of q_i in $\text{Up}(g_{i-1})$ satisfy $u < q_i$ by Invariant 1 for $\text{Up}(g_{i-1})$. The remaining descendants have an ancestor on the mid-trail between α and q_i in the new tree $\text{Up}(g_i)$. The window spanned by this ancestor and its birth has x in its out-panel and is thus on L_{dn}.

The interchange of minima does not modify $\mathsf{mid}(\alpha)$, but does modify $\mathsf{in}(\alpha)$. More precisely, $\mathsf{in}(\alpha) = q_i$ after the interchange. By assumption 4 and lemma 5.4 it follows for all $j \in [i+1, I]$ with $(p_j, q_j) \in L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$ that $f(q_j) < f(\mathsf{in}(\alpha))$ and $f(q_j) < f(\mathsf{mid}(\alpha))$. By lemma 5.4 and for all $j \in [i+1, I]$ with $(p_j, q_j) \in L_{\text{up}} \cup R_{\text{up}} \cup M_{\text{up}}$ it holds that $f(q_j) < f(q_i)$, which implies $f(q_j) < f(\mathsf{in}(\alpha))$. The value ε is defined to be smaller than the difference between the values of any two items, and thus there is no item with value in $[f(p_i), f(\alpha)]$. Because x is in the out-panel of $W(p_i, ,)q_i$ it holds that $f(p_i) < f(x)$ and thus $f(\alpha) < f(x)$.

For the sixth claim we first prove that $Stack_{i+1} \neq R_{\text{dn}}$. If $Stack_{i+1} \in R_{\text{dn}}$, then $x < q_{i+1} < p_{i+1}$ and x is in the out-panel of $W(p_{i+1}, q_{i+1})$ by definition of R_{dn}. This implies that q_{i+1} cannot be on the same trail as q_i and in order for x to be in the out-panel of $W(p_{i+1}, q_{i+1})$ it must in fact be either in the in-trail between α and $\mathsf{dth}(\alpha)$, or in a right spine of the right subtree. The former is impossible, as the in-trail between α and $\mathsf{dth}(\alpha)$ is empty. Now assume that q_{i+1} is on a right trail in the right up-tree. Then $p' = \mathsf{low}(q_{i+1}) < q_{i+1}$ and $\mathsf{dth}(p')$ must be on the left spine of the right tree, as otherwise x would not be in the out-panel of $W(p_{i+1}, q_{i+1})$. This implies that $p' < x < \mathsf{dth}(p')$, which in turn implies that $(p', \mathsf{dth}(p')) \in M_{\text{up}}$. As M_{up} and R_{dn} are sorted and $f(p') < f(p_{i+1})$ by Invariant 2, this implies that $Stack_{i+1}$ cannot be R_{dn}. Thus, if $Stack_{i+1} \in \{L_{\text{dn}}, R_{\text{dn}}\}$, then $Stack_{i+1} = L_{\text{dn}}$.

In this case $p_{i+1} < q_{i+1} < x$, which implies that q_{i+1} must be in the left tree and thus in the same tree as α. $\square$

Theorem B.3 (Correctness of Split). *Given a list of m items and corresponding map f, and an item ℓ with $2 \leq \ell \leq m-1$,* Split *splits the up-tree* $\mathrm{Up}(f)$ *into two up-trees* $\mathrm{Up}(g)$ *and* $\mathrm{Up}(h)$, *where* $\mathrm{Up}(g)$ *contains nodes u with $u \leq \ell$ and* $\mathrm{Up}(h)$ *contains the nodes $u \geq \ell$.*

Proof. Recall that we defined x to be the midpoint between ℓ and $\ell + 1$. The proof is by induction over the iterations of the loop in Split, taking lemma B.1 as the base case and lemma B.2 as the induction step. This proves the invariants (i) and (ii) introduced in section 5.3, which imply the theorem. $\square$

B.2 Gluing Two Banana Trees

In this section we prove that the algorithm Glue presented in section 5.3 correctly glues two banana trees $\mathrm{Up}(g)$ and $\mathrm{Up}(h)$ into $\mathrm{Up}(f)$, where $f = g \cdot h$. First, we show how to pre-process the functions g and h to obtain the case where g ends in an up-type item on the right, h begins with a down-type item on the left and this endpoint of g has lower value than the endpoint of h. Write ℓ for the rightmost item of g and ℓ' for the leftmost item of h. We assume $f(\ell) < f(\ell')$. In the case $f(\ell) > f(\ell')$ we obtain the symmetric case to the one described in section section 5.3 and the algorithm is symmetric. There are four cases depending on whether ℓ and ℓ' are up-type or down-type items: (1) ℓ and ℓ' are up-type; (2) ℓ is up-type and ℓ' is down-type; (3) ℓ is down-type and ℓ' is up-type; (4) ℓ and ℓ' are down-type. Case (2) is the one described in section 5.3 and we show how to reduce the other cases to this case.

(1) → (2): ℓ' becomes non-critical in $g \cdot h$ and we replace it by the dummy leaf α.

(3) → (2): ℓ and ℓ' become non-critical in $g \cdot h$. Since ℓ is down-type it is paired with a hook in $\mathrm{Up}(g)$ and we simply remove it along with the hook. As in the previous case we replace ℓ' by the dummy leaf α.

(4) → (2): ℓ becomes non-critical in $g \cdot h$. As in the previous ℓ is paired with a hook in $\mathrm{Up}(g)$ and we remove it along with the hook.

Whenever ℓ' is a down-type item it is paired with a hook and we replace the hook by the dummy leaf α. In all cases we begin with the situation described in section 5.3.

For the remainder of the section we assume that we are in case (2), and we now prove that the algorithm Glue yields the tree $\mathrm{Up}(f)$. The proof will be by induction over the iterations of the loop, with the base case and induction step in lemmas B.4 and B.5, respectively. We combine the two and show that the algorithm terminates correctly in theorem B.6. Write $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$ for the left and right tree after iteration i, respectively, with $\mathrm{Up}(g_0) = \mathrm{Up}(g)$ and $\mathrm{Up}(h_0) = \mathrm{Up}(h)$. Define α to be greater than the other nodes of $\mathrm{Up}(g_i)$ and less than the other nodes of $\mathrm{Up}(h_i)$. Furthermore, we write p_i, $\hat{p}_i$, q_i, q'_i for p, $\hat{p}$, q, q' in the i-th iteration.

Lemma B.4 (Base Case). *At the beginning of the first iteration it holds that*

1. $\mathrm{Up}(g_0)$ *and* $\mathrm{Up}(h_0)$ *satisfy Invariants 1 to 3,*
2. *the in-trail between α and* $\mathtt{dth}(\alpha)$ *is empty,* $\alpha \in \{\mathrm{Bth}(q_1), \mathrm{Bth}(q'_1)\}$, $f(\alpha) > f(\hat{p}_1)$, $f(\alpha) > f(p_1)$,

3. *α is the last node on the right spine of* $\mathrm{Up}(g_0)$ *or on the left spine of* $\mathrm{Up}(0)$*,* q_1 *is on the right spine of* $\mathrm{Up}(g_0)$ *and* q_1' *is on the left spine of* $\mathrm{Up}(h_0)$

4. *for all nodes u in* $\mathrm{Up}(g_1)$ *and all nodes v in* $\mathrm{Up}(h_1)$ *we have $u < v$,*

5. $\textsc{UndoInjury}(p_1, q_1)$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1))$ *and* $\textsc{UndoInjury}(p_1, q_1')$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1'))$*,*

6. $\textsc{UndoFatality}(p_1, q_1)$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1'))$ *and* $\textsc{UndoFatality}(p_1, q_1')$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1))$*,*

7. $\textsc{UndoScare}(\hat{p}_1, q_1)$ *being called implies* $\mathrm{Bth}(q_1) = \alpha$*,* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1'))$ *and* $\textsc{UndoScare}(\hat{p}_1, q_1')$ *being called implies* $\mathrm{Bth}(q_1') = \alpha$*,* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1))$*.*

Proof. The trees $\mathrm{Up}(g_0)$ and $\mathrm{Up}(h_0)$ are the input to the algorithm, i.e., $\mathrm{Up}(g)$ and $\mathrm{Up}(h)$, and thus satisfy Invariants 1 to 3. Since α is a hook, the in-trail between α and $\mathsf{dth}(\alpha)$ is empty. By definition $\alpha = \mathrm{Bth}(b_0') = \mathrm{Bth}(q_1')$. The inequalities $f(\alpha) > f(p_i)$ and $f(\alpha) > f(\hat{p}_1)$ also follow from the assumptions on g and h. The nodes $q_1 = b_0$ and $q_1' = b_0'$ are on the right spine of $\mathrm{Up}(g_0)$ and the left spine of $\mathrm{Up}(h_0)$, respectively. The node $\alpha = \mathrm{Bth}(q_1')$ is thus also on the spine of $\mathrm{Up}(h_0)$ and since it is a leaf it is the last node on the spine. Claim 4 holds by the assumption that items in g are all less than items in h.

We now prove claims 5, 6 and 7 If $f(q_1) < f(q_1)'$, then $p_1 = a_0$ and by Invariant 2 $f(p_1) > f(\hat{p}_1)$. Thus $\textsc{UndoInjury}(p_1, q_1)$ is executed. Since $\mathrm{Bth}(q_1) = a_0$ is an up-type item it also holds that $\mathsf{in}(q_1) = a_0$. As by definition $f(\alpha) > f(a_0)$ the inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1))$ holds. This implies claim 5. If $f(q_1) > f(q_1')$, then $p_1 = a_0$ and $p_1 \neq \mathrm{Bth}(q_1')$. Either $\textsc{UndoFatality}(p_1, q_1')$ or $\textsc{UndoScare}(\hat{p}_1, q_1)$ is executed. The inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_1))$ holds as we have just shown and this proves claims 6 and 7.

We have shown that all claims hold at the beginning of the first iteration and this proves the lemma. □

Lemma B.5 (Inductive Step)**.** *If at the beginning of the i-th iteration the following conditions hold*

1. $\mathrm{Up}(g_{i-1})$ *and* $\mathrm{Up}(h_{i-1})$ *satisfy Invariants 1 to 3,*

2. *the in-trail between α and* $\mathsf{dth}(\alpha)$ *is empty,* $\alpha \in \{\mathrm{Bth}(q_i), \mathrm{Bth}(q_i')\}$*,* $f(\alpha) > f(\hat{p}_i)$*,* $f(\alpha) > f(p_i)$*,*

3. *α is the last node on the right spine of* $\mathrm{Up}(g_{i-1})$ *or on the left spine of* $\mathrm{Up}(h_{i-1})$*,* q_i *is on the right spine of* $\mathrm{Up}(g_{i-1})$ *and* q_i' *is on the left spine of* $\mathrm{Up}(h_{i-1})$

4. *for all nodes u in* $\mathrm{Up}(g_i)$ *and all nodes v in* $\mathrm{Up}(h_i)$ *we have $u < v$,*

5. $\textsc{UndoInjury}(p_i, q_i)$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$ *and* $\textsc{UndoInjury}(p_i, q_i')$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i'))$*,*

6. $\textsc{UndoFatality}(p_i, q_i)$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i'))$ *and* $\textsc{UndoFatality}(p_i, q_i')$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$*,*

7. $\textsc{UndoScare}(\hat{p}_i, q_i)$ *being called implies* $\mathrm{Bth}(q_i) = \alpha$*,* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i'))$ *and* $\textsc{UndoScare}(\hat{p}_i, q_i')$ *being called implies* $\mathrm{Bth}(q_i') = \alpha$*,* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$*,*

then at the beginning of the next iteration, if it exists, it holds that

1. $\mathrm{Up}(g_i)$ *and* $\mathrm{Up}(h_i)$ *satisfy Invariants 1 to 3,*
2. *the in-trail between* α *and* $\mathsf{dth}(\alpha)$ *is empty,* $\alpha \in \{\mathrm{Bth}(q_{i+1}), \mathrm{Bth}(q'_{i+1})\}$*,* $f(\alpha) > f(\hat{p}_{i+1})$*,* $f(\alpha) > f(p_{i+1})$*,*
3. α *is the last node on the right spine of* $\mathrm{Up}(g_i)$ *or on the left spine of* $\mathrm{Up}(h_i)$*,* q_{i+1} *is on the right spine of* $\mathrm{Up}(g_i)$ *and* q'_{i+1} *is on the left spine of* $\mathrm{Up}(h_i)$
4. *for all nodes* u *in* $\mathrm{Up}(g_{i+1})$ *and all nodes* v *in* $\mathrm{Up}(h_{i+1})$ *we have* $u < v$*,*
5. $\textsc{UndoInjury}(p_{i+1}, q_{i+1})$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}))$ *and* $\textsc{UndoInjury}(p_{i+1}, q'_{i+1})$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q'_{i+1}))$*,*
6. $\textsc{UndoFatality}(p_{i+1}, q_{i+1})$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q'_{i+1}))$ *and* $\textsc{UndoFatality}(p_{i+1}, q'_{i+1})$ *being called implies* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}))$*,*
7. $\textsc{UndoScare}(\hat{p}_{i+1}, q_{i+1})$ *being called implies* $\mathrm{Bth}(q_{i+1}) = \alpha$*,* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q'_{i+1}))$ *and* $\textsc{UndoScare}(\hat{p}_{i+1}, q'_{i+1})$ *being called implies* $\mathrm{Bth}(q'_{i+1}) = \alpha$*,* $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}))$*.*

Proof. Assume that the conditions hold. We prove the claims for $f(q_i) < f(q'_i)$ by considering separately the three cases that the algorithm distinguishes. The case $f(q_i) > f(q'_i)$ is symmetric.

Case 1 ($f(p_i) < f(\hat{p}_i)$ and $p_i = \mathrm{Bth}(q_i)$): $\textsc{UndoInjury}(p_i, q_i)$ is called, which removes nodes j with $f(j) < f(q_i)$ from the mid-trail between α and $\mathsf{dth}(\alpha)$ and inserts them into the in-trail between p_i and q_i, specifically between the nodes $\mathsf{in}(q_i)$ and q_i.

We first prove that $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$ satisfy Invariants 1 to 3. By assumption 1 these invariants holds for $\mathrm{Up}(g_{i-1})$ and $\mathrm{Up}(h_{i-1})$. To see that they also hold for $\mathrm{Up}(h_i)$, observe that removing maxima along with its banana subtree does not affect the invariants. For $\mathrm{Up}(g_i)$ we first show Invariant 3. Write j^- and j^+ for the lowest and highest of the nodes are moved, respectively. That is, in $\mathrm{Up}(g_{i-1})$ $j^- = \mathsf{mid}(\alpha)$ and j^+ is the highest node on the mid-trail between α and $\mathsf{dth}(\alpha)$ such that $f(j^+) < f(q_i)$. Write j_q for $\mathsf{in}(q_i)$ in $\mathrm{Up}(g_{i-1})$. In $\mathrm{Up}(g_i)$ these nodes are inserted such that $j^- = \mathsf{up}(j_q)$ and $j^+ = \mathsf{in}(q_i)$. By the assumption that $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$ (assumption 5) we thus have $f(\mathsf{dn}(j^-)) < f(j^-)$ in $\mathrm{Up}(g_i)$. Furthermore, $f(j^+) < f(\mathsf{up}(j^+)) = f(q_i)$. By assumption 4 and Invariant 3 for $\mathrm{Up}(h_{i-1})$ it holds in $\mathrm{Up}(g_i)$ that $\mathsf{dn}(j^-) < j^- < \mathsf{up}(j^-) < \cdots < \mathsf{dn}(j^+) < j^+$ and $\mathsf{up}(j^+) = q_i < j^+$. Thus Invariant 3 holds in $\mathrm{Up}(g_i)$. To see that Invariant 1 is satisfied note that the j that are inserted into $\mathrm{Up}(g_i)$ satisfy $q_i < j$ by assumption 4. Since q_i is on the right spine $q_i < p_i = \mathrm{Bth}(q_i)$ and thus the condition of Invariant 1 holds for q_i and its ancestors. The nodes j gain descendants u in the same subtree $\mathsf{dn}(j)$, and these nodes u satisfy $u < j$, again by assumption 4. Thus, these nodes j also satisfy the condition for Invariant 1. For no other node do the subtrees change and thus Invariant 1 holds in $\mathrm{Up}(g_i)$. Finally, we prove Invariant 2 by analyzing the nodes for which the nodes at $\mathsf{dth}(\mathsf{low}(\cdot))$ changes. These are the nodes v with $\mathsf{dth}(v) = j$ for some moved node j. They originally have $\mathsf{dth}(\mathsf{low}(v)) = \alpha$ and this changes to $\mathsf{dth}(\mathsf{low}(v)) = p_i$. Assumption 2 that $f(\alpha) > f(p_i)$ thus implies $f(\mathsf{dth}(\mathsf{low}(v))) > f(\alpha) > f(p_i)$. This implies Invariant 2. In the remainder of the proof for this case we assume Invariants 1 to 3 for $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$.

The in-trail between α and $\mathsf{dth}(\alpha)$ is empty at the beginning of the i-th iteration, is not modified by $\textsc{UndoInjury}$ and is thus also empty at the beginning of the next iteration. As $q'_{i+1} = q'_i$ and $\mathsf{dth}(\alpha) = q'_i$ is unchanged it holds that $\alpha = \mathrm{Bth}(q'_{i+1})$ at the beginning of the next iteration. If in the next iteration $f(q_{i+1}) < f(q'_{i+1})$, then $p_{i+1} = \mathrm{Bth}(q_{i+1}) = \mathsf{low}(q_i) = \hat{p}_i$ and $\hat{p}_{i+1} = \mathsf{low}(q_{i+1})$. By Invariant 2 this implies $f(\hat{p}_{i+1}) < f(p_{i+1}) < f(p_i) < \alpha$, where the last inequality follows from assumption 2. Otherwise, if in the next iteration $f(q_{i+1}) > f(q'_{i+1})$, then $p_{i+1} = \mathrm{Bth}(q_{i+1}) = \mathsf{low}(q_i)$ (since $\mathrm{Bth}(q'_{i+1}) = \alpha$) and $\hat{p}_{i+1} = \mathsf{low}(q'_{i+1}) = \mathsf{low}(q'_i)$. By Invariant 2 and assumption 2 this implies $f(\hat{p}_{i+1}) < f(\alpha)$ and $f(p_{i+1}) < f(p_i) < f(\alpha)$. This proves claim 2. Claim 3 follows immediately from the assumption 3: α was the last node on the left spine of $\mathrm{Up}(h_{i-1})$ at the beginning of the iteration, and the removal of nodes from its mid-trail does not change that; $q_{i+1} = \mathsf{dth}(\mathsf{low}(q_i))$, which is on the right spine of $\mathrm{Up}(g_i)$ since q_i is on the right spine of $\mathrm{Up}(g_{i-1})$ and $q'_{i+1} = q'_i$ is on the left spine of $\mathrm{Up}(h_i)$ because q'_i is on the left spine of $\mathrm{Up}(h_{i-1})$.

Since α is the last node on the left spine of $\mathrm{Up}(h_{i-1})$ and its in-trail is empty, the nodes that are moved to $\mathrm{Up}(g_i)$ are the leftmost nodes of $\mathrm{Up}(h_{i-1})$ other than α. Assumption 4 and the definition of α thus imply the claim 4.

We now prove claims 5, 6 and 7. Recall that $q'_{i+1} = q_{i+1}$ and $q'_{i+1} = \mathsf{dth}(\mathsf{low}(q_i))$. In total there are six cases, but only three can occur:

(i) $f(q_{i+1}) < f(q'_{i+1})$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1})$
$\implies$ call to UndoInjury(p_{i+1}, q_{i+1})

(ii) $f(q_{i+1}) > f(q'_{i+1})$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1})$
$\implies$ call to UndoFatality(p_{i+1}, q'_{i+1})

(iii) $f(q_{i+1}) > f(q'_{i+1})$, $f(\hat{p}_{i+1}) > f(p_{i+1})$
$\implies$ call to UndoScare$(\hat{p}_{i+1}, q'_{i+1})$

Since $p_{i+1} = \mathrm{Bth}(q_{i+1}) \neq \alpha$, if $f(q_{i+1}) < f(q_{i+1})'$, then UndoFatality will not be called. By Invariant 2 it also holds that $f(\hat{p}_{i+1}) < f(p_{i+1})$, which implies that UndoScare will not be called. Finally, $\mathrm{Bth}(q'_{i+1}) = \alpha$ implies that if $f(q_{i+1}) > f(q'_{i+1})$, then UndoInjury will not be called. We analyze the three possible cases separately:

(i) $f(q_{i+1}) < f(q'_{i+1})$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1})$ and UndoInjury(p_{i+1}, q_{i+1}) is called. We have shown above that q_{i+1} is on the spine, as is q_i, and thus $q_{i+1} = \mathsf{dth}(\mathsf{low}(q_i))$ implies $\mathsf{in}(q_{i+1}) = q_i$. Since $f(\mathsf{mid}(\alpha))$ was not moved to $\mathrm{Up}(g_i)$ and $\mathsf{dth}(\alpha) = f(q'_i) > f(q_i)$ it follows that $f(\mathsf{mid}(\alpha)) > f(q_i) = f(\mathsf{in}(q_{i+1}))$.

(ii) $f(q_{i+1}) > f(q'_{i+1})$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1})$ and UndoFatality(p_{i+1}, q'_{i+1}) is called. The inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}))$ holds by the same argument as in the previous case.

(iii) $f(q_{i+1}) > f(q'_{i+1})$, $f(\hat{p}_{i+1}) > f(p_{i+1})$ and UndoScare$(\hat{p}_{i+1}, q'_{i+1})$ is called. Since $q'_{i+1} = q'_i$ and $\alpha = \mathrm{Bth}(q'_i)$ it holds that $\alpha = \mathrm{Bth}(q'_{i+1})$. The inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}))$ holds by the same argument as in the other cases.

In all three cases the claims hold.

Case 2 ($f(p_i) < f(\hat{p}_i)$ and $p_i = \mathrm{Bth}(q'_i)$): UndoFatality(p_i, q_i) is called, which takes on the in-trail between p_i and q'_i and the nodes j with $f(j) < f(q_i)$ on the mid-trail between p_i and q'_i and swaps them with $\alpha = \mathrm{Bth}(q_i)$, such that the nodes formerly on the in-trail extend the mid-trail of q_i and the nodes j extend the in-trail of q_i.

We first prove that $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$ satisfy Invariants 1 to 3. By assumption 1 these invariants hold for $\mathrm{Up}(g_{i-1})$ and $\mathrm{Up}(h_{i-1})$. To see that they also hold for $\mathrm{Up}(h_i)$, observe that replacing a part of the banana (p_i, q'_i) by α does not affect Invariants 1 and 3. By assumption 2 $f(\alpha) > f(p_i)$ and thus $f(\alpha) > f(p_i) > f(\mathsf{low}(\mathsf{dth}(\alpha))$, where the second inequality follows from Invariant 2 for $\mathrm{Up}(h_{i-1})$. As $\mathsf{low}(\mathsf{dth}(\cdot))$ changes for no other node Invariant 2 also holds for $\mathrm{Up}(h_i)$. Write j^+

for the topmost node on the mid-trail between p_i and q_i' in $\mathrm{Up}(h_{i-1})$ that is moved to $\mathrm{Up}(g_i)$, and k^+ for the topmost node on the in-trail between p_i and q_i' in $\mathrm{Up}(h_{i-1})$. As the entire in-trail is moved to the left tree $k^+ = \mathsf{in}(q_i')$ in $\mathrm{Up}(h_{i-1})$. Write m_α for the node $\mathsf{mid}(\alpha)$ in $\mathrm{Up}(g_{i-1})$. In $\mathrm{Up}(g_i)$ $\mathsf{dn}(m_\alpha) = k^+$. By assumption 6 we have $f(m_\alpha) > f(k^+)$, and by definition of j^+ we have $f(j^+) < f(q_i)$. Furthermore, by assumption 4 it holds that $m_\alpha < k^+$ and $q_i < j^+$. Since these are the only node where $\mathsf{up}(\cdot)$ and $\mathsf{dn}(\cdot)$ pointers change it follows that Invariant 3 holds in $\mathrm{Up}(g_i)$. To see that Invariant 1 is satisfied note that the nodes u that are inserted into $\mathrm{Up}(g_i)$ satisfy $u > v$ for any node v that is already in this tree from the previous iteration. With $q_i < \mathrm{Bth}(q_i)$ as q_i is on the right spine and $s < \mathsf{dn}(s)$ for any node on the mid-trail beginning at q_i Invariant 1 follows. The nodes for which $\mathsf{low}(\mathsf{dth}(\cdot))$ changes are those nodes t with $\mathsf{dth}(t)$ on the mid-trail beginning at q_i and the node p_i. It changes from α to p_i. By assumption 2 $f(p_i) < f(\alpha)$, so $f(\mathsf{low}(\mathsf{dth}(t))) > f(\alpha) > f(p_i)$. Furthermore, for p_i we have $\mathsf{low}(\mathsf{dth}(p_i)) = \hat{p}_i$ and they satisfy $f(p_i) > f(\hat{p})$. This implies Invariant 2 for $\mathrm{Up}(g_i)$ and concludes the proof of claim 1. In the remainder of the proof for this case we assume Invariants 1 to 3 for $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$.

After the iteration $\mathsf{dth}(\alpha) = q_i'$. As the nodes on the in-trail beginning at q_i' are removed, this trail is empty. Since $q_{i+1}' = q_i'$ it holds that $\alpha = \mathrm{Bth}(q_{i+1}')$. For the inequalities $f(\alpha) > f(p_{i+1})$ and $f(\alpha) > f(\hat{p}_{i+1})$ we refer to the proof for the previous case. It follows that claim 2 holds. The node q_i' is on the left spine of $\mathrm{Up}(h_{i-1})$ and thus $q_{i+1}' = q_i'$ is on the left spine of $\mathrm{Up}(h_i)$. Since $\alpha = \mathrm{Bth}(q_{i+1}')$ in $\mathrm{Up}(h_i)$ it also follows that α is on the left spine of $\mathrm{Up}(h_i)$. Similarly, in $\mathrm{Up}(g_{i-1})$ and $\mathrm{Up}(g_i)$ the node q_i is on the right spine and thus $q_{i+1} = \mathsf{dth}(\mathsf{low}(q_i))$ is also on the right spine. This proves claim 3. For the proof of claim 4 we again refer to the previous case, since the proof is similar.

We now prove claims 5, 6 and 7. Again, in the next iteration only three out of six cases can occur:

(i) $f(q_{i+1}) < f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p = \mathrm{Bth}(q_{i+1})$
$\implies$ call to UndoInjury(p_{i+1}, q_{i+1})

(ii) $f(q_{i+1}) > f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p = \mathrm{Bth}(q_{i+1})$
$\implies$ call to UndoFatality(p_{i+1}, q_{i+1}')

(iii) $f(q_{i+1}) > f(q_{i+1}')$, $f(\hat{p}_{i+1}) > f(p_{i+1})$
$\implies$ call to UndoScare$(\hat{p}_{i+1}, q_{i+1}')$

Note that these are the same cases as in Case 1. The proof that the remaining three cases cannot occur is identical. The remainder of the proof is also similar to the proof in case 1:

(i) $f(q_{i+1}) < f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p = \mathrm{Bth}(q_{i+1})$ and UndoInjury(p_{i+1}, q_{i+1}) is called. Again, q_{i+1} and q_i are on the spine and thus $\mathsf{in}(q_{i+1}) = q_i$. $f(\mathsf{mid}(\alpha))$ is one of the nodes that have not been moved from the mid-trail, and thus $f(\mathsf{mid}(\alpha)) > f(q_i) = f(\mathsf{in}(q_{i+1}))$.

(ii) $f(q_{i+1}) > f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p = \mathrm{Bth}(q_{i+1})$ and UndoFatality(p_{i+1}, q_{i+1}') is called. The inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}))$ holds by the same argument as in the previous case.

(iii) $f(q_{i+1}) > f(q_{i+1}')$, $f(\hat{p}_{i+1}) > f(p_{i+1})$ and UndoScare$(\hat{p}_{i+1}, q_{i+1}')$ is called. As before $\alpha = \mathrm{Bth}(q_i') = \mathrm{Bth}(q_{i+1}')$. The inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1})$ holds as shown in the other cases.

This proves claims 5, 6 and 7.

Case 3 ($f(\hat{p}_i) < f(p_i)$): UndoScare$(\hat{p}_i, q_i)$ is called, which sets $f(\alpha) = f(\hat{p}_i) - \varepsilon$ and then performs an interchange of minima. Note that $\alpha = \mathrm{Bth}(q_i)$, as $f(p_i) < f(\hat{p}_i)$, which by Invariant 2 for $\mathrm{Up}(g_i)$ cannot be if $p_i = \mathrm{Bth}(q_i)$. Thus, $\alpha \neq \mathrm{Bth}(q_i')$ and by assumption 2 this implies $\alpha = \mathrm{Bth}(q_i)$. Claim 4 follows directly from assumption 4, as the contents of the trees do not change. Invariants 1 to 3 immediately follow for $\mathrm{Up}(g_i)$ from the correctness of the interchange of minima. As $\mathrm{Up}(h_i) = \mathrm{Up}(h_{i-1})$ these invariants hold $\mathrm{Up}(h_i)$ by assumption 1. In the remainder of the proof for this case we assume Invariants 1 to 3 for $\mathrm{Up}(g_i)$ and $\mathrm{Up}(h_i)$.

The in-trail between α and $\mathsf{dth}(\alpha)$ is empty in $\mathrm{Up}(g_{i-1})$ by assumption 2. As $\mathsf{dth}(\alpha) = q_i$ is on a spine the in-trail remains empty after the interchange of minima. The interchange also results in $\mathsf{dth}(\alpha) = q_{i+1}$ in $\mathrm{Up}(g_i)$. If $f(q_{i+1}) < f(q_{i+1})'$, then $p_{i+1} = \mathrm{Bth}(q_{i+1}') = \mathrm{Bth}(q_i') = p_i$. As $f(p_i) < f(\hat{p}_i)$ and ε is defined to be sufficiently small it follows that $f(p_{i+1}) < f(\alpha)$. Furthermore, $f(\hat{p}_{i+1}) < f(\alpha)$ by Invariant 2. Otherwise, if $f(q_{i+1}) > f(q_{i+1})'$, then $p_{i+1} = \mathrm{Bth}(q_{i+1}') = \mathrm{Bth}(q_i') = p_i$, which we have just shown to satisfy $f(p_{i+1}) < f(\alpha)$. By Invariant 2 $f(\hat{p}_{i+1}) < f(p_{i+1})$, which implies $f(\hat{p}_{i+1}) < f(\alpha)$. This proves claim 2.

The node q_{i+1} is on the spine since it is already on the spine in $\mathrm{Up}(g_{i-1})$ and the interchange of minima does not change its ancestors. The node α remains on the spine since its in-trail remains empty and $q_{i+1} = \mathsf{dth}(\alpha)$ is on the left spine in $\mathrm{Up}(g_{i+1})$. Since it is also a leaf it is the last node on the spine. Finally, q_{i+1}' is on the spine since $q_{i+1}' = q_i'$. This proves 3.

It remains to prove claims 5, 6 and 7. Again, in the next iteration only three out of six cases can occur:

(i) $f(q_{i+1}) < f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1}')$
$\implies$ call to UndoFatality(p_{i+1}, q_{i+1})

(ii) $f(q_{i+1}) < f(q_{i+1}')$, $f(\hat{p}_{i+1}) > f(p_{i+1})$
$\implies$ call to UndoScare$(\hat{p}_{i+1}, q_{i+1})$

(iii) $f(q_{i+1}) > f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i_1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1}')$
$\implies$ call to UndoInjury(p_{i+1}, q_{i+1}')

Since $\mathrm{Bth}(q_{i+1}) = \alpha$ we cannot have $p_{i+1} = \mathrm{Bth}(q_{i+1})$ and thus if $f(q_{i+1}) < f(q_{i+1}')$ then UndoInjury is not called. If $f(q_{i+1}) > f(q_{i+1}')$ then $f(p_{i+1}) > f(\hat{p}_{i+1})$ by Invariant 2, and so UndoScare is not called. UndoFatality is not called in this case as $p_{i+1} = \mathrm{Bth}(q_{i+1}')$. We now analyze the three possible cases:

(i) $f(q_{i+1}) < f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i+1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1}')$ and UndoFatality(p_{i+1}, q_{i+1}) is called. The node $\mathsf{mid}(\alpha)$ is the same in $\mathrm{Up}(g_{i-1})$ and $\mathrm{Up}(g_i)$. Furthermore $\mathsf{in}(q_{i+1})' = \mathsf{in}(q_i')$, as $\mathrm{Up}(h_i) = \mathrm{Up}(h_{i-1})$. The assumption that $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i')$ thus implies $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}'))$.

(ii) $f(q_{i+1}) < f(q_{i+1}')$, $f(\hat{p}_{i+1}) > f(p_{i+1})$ and UndoScare$(\hat{p}_{i+1}, q_{i+1})$ is called. We have shown above that $\mathsf{dth}(\alpha) = q_{i+1}$ which implies $\alpha = \mathrm{Bth}(q_{i+1})$.

(iii) $f(q_{i+1}) > f(q_{i+1}')$, $f(p_{i+1}) > f(\hat{p}_{i_1})$ and $p_{i+1} = \mathrm{Bth}(q_{i+1}')$ and UndoInjury(p_{i+1}, q_{i+1}') is called. The inequality $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_{i+1}'))$ holds by the same argument as in the first case.

□

Theorem B.6 (Correctness of Glue)**.** *Given two lists of items and corresponding maps g and h* Glue *applied to* $\mathrm{Up}(g)$ *and* $\mathrm{Up}(h)$ *yields the up-tree* $\mathrm{Up}(f) = \mathrm{Up}(g \cdot h)$*.*

Proof. We show by induction over the iterations of Glue that at the beginning of the i-th iteration it holds that

1. $\mathrm{Up}(g_{i-1})$ and $\mathrm{Up}(h_{i-1})$ satisfy Invariants 1 to 3,
2. the in-trail between α and $\mathsf{dth}(\alpha)$ is empty, $\alpha \in \{\mathrm{Bth}(q_i), \mathrm{Bth}(q_i')\}$, $f(\alpha) > f(\hat{p}_i)$, $f(\alpha) > f(p_i)$,
3. α is the last node on the right spine of $\mathrm{Up}(g_{i-1})$ or on the left spine of $\mathrm{Up}(h_{i-1})$, q_i is on the right spine of $\mathrm{Up}(g_{i-1})$ and q_i' is on the left spine of $\mathrm{Up}(h_{i-1})$
4. for all nodes u in $\mathrm{Up}(g_i)$ and all nodes v in $\mathrm{Up}(h_i)$ we have $u < v$,
5. UndoInjury(p_i, q_i) being called implies $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$ and UndoInjury(p_i, q_i') being called implies $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i'))$,
6. UndoFatality(p_i, q_i) being called implies $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i'))$ and UndoFatality(p_i, q_i') being called implies $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$,
7. UndoScare$(\hat{p}_i, q_i)$ being called implies $\mathrm{Bth}(q_i) = \alpha$, $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i'))$ and UndoScare$(\hat{p}_i, q_i')$ being called implies $\mathrm{Bth}(q_i') = \alpha$, $f(\mathsf{mid}(\alpha)) > f(\mathsf{in}(q_i))$.

Lemma B.4 proves the base case for the first iteration $i = 1$, and the induction step is in lemma B.5. We assume these claims for all iterations.

We now show that there exists an iteration after which one tree is empty. Assume that the global maximum of h has greater value than the global maximum of g and both trees have nodes other than the special root and α. The case where the global maximum of g has greater value than that of h is symmetric. Consider any iteration I in which $f(q_I') > f(u)$ for any $u \neq \beta_g$ in $\mathrm{Up}(g_{I-1})$ and $q_I = \beta_g$. Since in some iteration $q_I' = \beta_h$ this iteration exsists. In this iteration $f(q_I') < f(q_I)$, either since q_I is a special root and q_I' is not, or since $f(\beta_h) < f(\beta_g)$ by definition. We distinguish two cases: $q_I' \neq \beta_h$ and $q_I = \beta_h$.

$q_I' \neq \beta_h$: There are three cases to consider: (1) $\mathrm{Bth}(q_I') \neq \alpha$ and $p_I = \mathrm{Bth}(q_I')$; (2) $\mathrm{Bth}(q_I') = \alpha$ and $f(p_I) > f(\hat{p}_I)$; (3) $\mathrm{Bth}(q_I') = \alpha$ and $f(\hat{p}_I) > f(p_I)$. In case (1) $f(p_I) > f(\hat{p}_I)$ by Invariant 2 and thus UndoInjury(p_I, q_I') is called. In case (2) UndoFatality(p_I, q_I') is called. In case (3) UndoScare$(\hat{p}_I, q_I')$ is called. If case (1) occurs, then $\alpha = \mathrm{Bth}(q_I) = \mathrm{Bth}(\beta_g)$, the in-trail between α and β_g is empty, and all nodes from the mid-trail between α and β_g are moved to $\mathrm{Up}(h_I)$. This leaves $\mathsf{in}(\beta_g) = \mathsf{mid}(\beta_g) = \alpha$ and thus the loop terminates. If case (2) occurs, then $p_I = \mathrm{Bth}(q_I) = \mathrm{Bth}(\beta_g)$ and all nodes on trails between p_I and β_g except β_g are moved to $\mathrm{Up}(h_I)$, and α is moved to $\mathrm{Up}(g_I)$ such that $\mathsf{in}(\beta_g) = \mathsf{mid}(\beta_g) = \alpha$. Thus, the loop terminates. In case (3) $f(\alpha)$ is set to $f(\hat{p}_I) - \varepsilon$ and an interchange of minima between α and $\hat{p}_I$ is performed. Thus, both trees are not empty. However, $q_{I+1}' = \mathsf{up}(q_I)$ is the next node upwards on the spine towards β_h and the next iteration is another iteration where $f(q_I') > f(u)$ for all $u \neq \beta_g \in \mathrm{Up}(g_i)$.

$q_I' = \beta_h$: This case is eventually reached, since the q_i' move upwards on the spine to β_h. We have defined $f(\beta_h) < f(\beta_g)$. There are again three cases to consider: (1) $\mathrm{Bth}(q_I') \neq \alpha$ and $p_I = \mathrm{Bth}(q_I')$; (2) $\mathrm{Bth}(q_I') = \alpha$ and $f(p_I) > f(\hat{p}_I)$; (3) $\mathrm{Bth}(q_I') = \alpha$ and $f(\hat{p}_I) > f(p_I)$. These are the same cases as above and in case (1) and (2) the loop terminates with $\mathsf{in}(\beta_g) = \mathsf{mid}(\beta_g) = \alpha$, as above. The third case, where $f(\hat{p}_I) > f(p_I)$, cannot occur since we defined $f(\hat{p}_I) = f(\mathsf{low}(\beta_h)) = f(\mathtt{nil}) = -\infty$. Thus, when $q_I' = \beta_h$, the algorithm terminates.

The choice that $f(\beta_h) < f(\beta_g)$ is arbitrary and we could have also chosen $f(\beta_g) < f(\beta_h)$. In this case we would get three symmetric cases for $q'_I = \beta_h$ with the same outcome.

Once the loop terminates after iteration I^* with $\mathsf{in}(\beta_g) = \mathsf{mid}(\beta_g) = \alpha$, the tree $\mathrm{Up}(h_{I^*})$ contains all items from $\mathrm{Up}(g)$ and $\mathrm{Up}(h)$ except for the dummy α. Thus, $\mathrm{Up}(h_{I^*}) = \mathrm{Up}(g \cdot h)$, which concludes the proof. □

References

[1] C.R. Aragon and R.G. Seidel. Randomized search trees. *Algorithmica* **16** (1996), 464–497.

[2] R. Biswas, S. Cultrera di Montesano, H. Edelsbrunner and M. Saghafian. Geometric characterization of the persistence of 1D maps. *J. Appl. Comput. Topol.*, (2023), doi.org/10.100/ s41468-023-00126-9.

[3] G. Carlsson. Topology and data. *Bull. Amer. Math. Soc.* **46** (2009), 255–308.

[4] Y.M. Chung, C.S. Hu, Y.L. Lo and H.T. Wu. A persistent homology approach to heart rate variability analysis with an application to sleep-walk classification. *Frontier Physiol.* **12** (2021), 637684.

[5] D. Cohen-Steiner, H. Edelsbrunner and J. Harer. Extending persistence using Poincaré and Lefschetz duality. *Found. Comput. Math.* **9** (2009), 79–103. Erratum 133–134.

[6] D. Cohen-Steiner, H. Edelsbrunner and D. Morozov. Vines and vineyards by updating persistence in linear time. *In* "Proc. 22nd Ann. Sympos. Comput. Geom., 2006", 119-126.

[7] M.-L. Dequeant, S. Ahnert, H. Edelsbrunner, T.M.A. Fink, E.F. Glynn, G. Hattem, A. Kudlicki, Y. Mileyko, J. Morton, A.R. Mushegian, L. Pachter, M. Rowicka, A. Shiu, B. Sturmfels and O. Pourquie. Comparison of pattern detection methods in microarray time series of the segmentation clock. *PLoS One* **3** (2008), e2856, doi:10.1371/journal.pone.0002856.

[8] T.K. Dey and T. Hou. Updating zigzag persistence and maintaining representatives over changing filtrations. `arXiv:2112.02352v1`, (2021).

[9] H. Edelsbrunner and J.L. Harer. *Computational Topology. An Introduction.* American Mathematical Society, Providence, Rhode Island, 2010.

[10] M. Gidea and Y.A. Katz. Topological data analysis of financial time series: landscapes of crashes. *Phys. A: Stat. Mech. Appl.* **491** (2018), 820-–834.

[11] M. Glisse. Fast persistent homology computation for functions on $\mathbb{R}$. `arXiv:2301.04745v1` (2023).

[12] G. Graff, B. Graff, P. Pilarczyk, G. Jabłoński, D. Gasecki and K. Narkiewicz. Persistent homology as a new method of the assessment of heart rate variability. *PLoS ONE* (2021), 0253851.

[13] Y. Luo and B.J. Nelson. Accelerating iterated persistent homology computations with warm starts. `arXiv:2108.05022` (2021).

[14] N. Otter, M.A. Porter, U. Tillmann, P. Grindrod and H.A. Harrington. A roadmap for the computation of persistent homology. *EPJ Data Science* **6** (2017), 1–38.

[15] J.A. Perea, A. Deckard, S.B. Haase and J. Harer. SW1PerS: sliding windows and 1-persistence scoring; discovering periodicity in gene expression time series data. *BMC Bioinformatics* **16** (2015), article #257.

[16] M. Piekenbrock and J.A. Perea. Move schedules: fast persistence computations in coarse dynamic settings. `arXiv:2104.12285v2` (2021).

[17] D. Smirnov and D. Morozov. Triplet merge trees. In *Topological Methods in Data Analysis and Visualization V* (TopoInVis '17), 1–17, 2019.

[18] J. Vuillemin. A unifying look at data structures. *Commun. ACM* **23** (1980), 229–239.