

Rollercoasters and Caterpillars

Therese Biedl¹

School of Computer Science, University of Waterloo, Canada
biedl@uwaterloo.ca

Ahmad Biniiaz²

School of Computer Science, University of Waterloo, Canada
ahmad.biniiaz@gmail.com

Robert Cummings

School of Computer Science, University of Waterloo, Canada
rcummings000@gmail.com

Anna Lubiw³

School of Computer Science, University of Waterloo, Canada
alubiw@uwaterloo.ca

Florin Manea⁴

Department of Computer Science, Kiel University, D-24098 Kiel, Germany
flm@zs.uni-kiel.de

Dirk Nowotka⁵

Department of Computer Science, Kiel University, D-24098 Kiel, Germany
dn@zs.uni-kiel.de

Jeffrey Shallit⁶

School of Computer Science, University of Waterloo, Canada
shallit@cs.uwaterloo.ca

Abstract

A *rollercoaster* is a sequence of real numbers for which every maximal contiguous subsequence – increasing or decreasing – has length at least three. By translating this sequence to a set of points in the plane, a rollercoaster can be defined as an x -monotone polygonal path for which every maximal sub-path, with positive- or negative-slope edges, has at least three vertices. Given a sequence of distinct real numbers, the rollercoaster problem asks for a maximum-length (not necessarily contiguous) subsequence that is a rollercoaster. It was conjectured that every sequence of n distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$ for $n > 7$, while the best known lower bound is $\Omega(n/\log n)$. In this paper we prove this conjecture. Our proof is constructive and implies a linear-time algorithm for computing a rollercoaster of this length. Extending the $O(n \log n)$ -time algorithm for computing a longest increasing subsequence, we show how to compute a maximum-length rollercoaster within the same time bound. A maximum-length rollercoaster in a permutation of $\{1, \dots, n\}$ can be computed in $O(n \log \log n)$ time.

The search for rollercoasters was motivated by orthogeodesic point-set embedding of caterpillars. A *caterpillar* is a tree such that deleting the leaves gives a path, called the *spine*. A *top-view caterpillar* is one of maximum degree 4 such that the two leaves adjacent to each vertex lie on opposite sides of the spine. As an application of our result on rollercoasters, we are able to

¹ Supported by NSERC.

² Supported by NSERC Postdoctoral Fellowship.

³ Supported by NSERC.

⁴ Supported by DFG.

⁵ Supported by DFG.

⁶ Supported by NSERC Grant # 105829/2013.



find a planar drawing of every n -vertex top-view caterpillar on every set of $\frac{25}{3}(n+4)$ points in the plane, such that each edge is an orthogonal path with one bend. This improves the previous best known upper bound on the number of required points, which is $O(n \log n)$. We also show that such a drawing can be obtained in linear time, when the points are given in sorted order.

2012 ACM Subject Classification Theory of computation → Algorithm design techniques

Keywords and phrases sequences, alternating runs, patterns in permutations, caterpillars

Digital Object Identifier 10.4230/LIPIcs.ICALP.2018.18

Related Version A full version of the paper is available at [3], <https://arxiv.org/abs/1801.08565>.

1 Introduction

A *run* in a sequence of real numbers is a maximal contiguous subsequence that is increasing or decreasing. A *rollercoaster* is a sequence of real numbers such that every run has length at least three.⁷ For example the sequence $(8, 5, 1, 3, 4, 7, 6, 2)$ is a rollercoaster with runs $(8, 5, 1)$, $(1, 3, 4, 7)$, $(7, 6, 2)$, which have lengths 3, 4, 3, respectively. The sequence $(8, 5, 1, 7, 6, 2, 3, 4)$ is not a rollercoaster because its run $(1, 7)$ has length 2. Given a sequence $S = (s_1, s_2, \dots, s_n)$ of n distinct real numbers, the rollercoaster problem is to find a maximum-size set of indices $i_1 < i_2 < \dots < i_k$ such that $(s_{i_1}, s_{i_2}, \dots, s_{i_k})$ is a rollercoaster. In other words, this problem asks for a longest rollercoaster in S , i.e., a longest subsequence of S that is a rollercoaster.

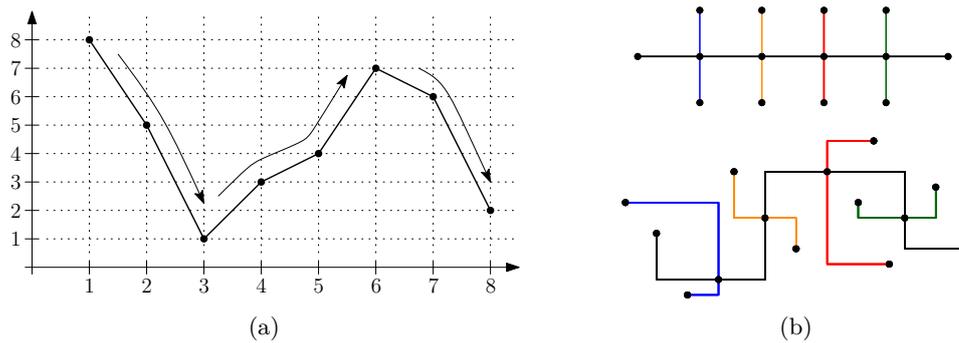
One can interpret S as a set P of points in the plane by translating each number $s_i \in S$ to a point $p_i = (i, s_i)$. With this translation, a rollercoaster in S translates to a “rollercoaster” in P , which is a polygonal path whose vertices are points of P and such that every maximal sub-path, with positive- or negative-slope edges, has at least three points. See Figure 1(a). Conversely, for any point set in the plane, the y -coordinates of the points, ordered by their x -coordinates, form a sequence of numbers. Therefore, any rollercoaster in P translates to a rollercoaster of the same length in S .

The best known lower bound on the length of a longest rollercoaster is $\Omega(n/\log n)$ due to Biedl et al. [4], who posed the following conjecture; see Appendix B in the full version of [4].

► **Conjecture 1.** *Every sequence of $n > 7$ distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$.*

Conjecture 1 can be viewed as a statement about patterns in permutations, a topic with a long history, and the subject of much current research. For example, the Eulerian polynomials, introduced by Euler in 1749, are the generating function for the number of descents in permutations. For surveys of recent work, see, for example, Linton et al. [13] and Kitaev [12]. Specifically, Conjecture 1 is related to the following seminal result of Erdős and Szekeres [7] in the sense that they prove the existence of an increasing or a decreasing subsequence of length at least $a + 1$ for $n = a^2 + 1$, which is essentially a rollercoaster with one run.

⁷ The term “rollercoaster permutation” has also been used to refer to a permutation that, together with all its subsequences, has maximum number of changes from increasing to decreasing and vice versa; see e.g. [1].



■ **Figure 1** (a) Translating the sequence $(8, 5, 1, 3, 4, 7, 6, 2)$ to a set of points. (b) A planar L-shaped drawing of a top-view caterpillar.

► **Theorem 2** (Erdős and Szekeres, 1935). *Every sequence of $ab + 1$ distinct real numbers contains an increasing subsequence of length at least $a + 1$ or a decreasing subsequence of length at least $b + 1$.*

Hammersley [11] gave an elegant proof of the Erdős–Szekeres theorem that is short, simple, and based on the pigeonhole principle. The Erdős–Szekeres theorem also follows from the well-known decomposition of Dilworth (see [17]). The following is a restatement of Dilworth’s decomposition for sequences of numbers.

► **Theorem 3** (Dilworth, 1950). *Any finite sequence S of distinct real numbers can be partitioned into k ascending sequences where k is the maximum length of a descending sequence in S .*

Besides its inherent interest, the study of rollercoasters is motivated by point-set embedding of caterpillars [4]. A *caterpillar* is a tree such that deleting the leaves gives a path, called the *spine*. An *ordered caterpillar* is a caterpillar in which the cyclic order of the edges incident to each vertex is specified. A *top-view caterpillar* is an ordered caterpillar where all vertices have degree 4 or 1 such that the two leaves adjacent to each spine vertex lie on opposite sides of the spine; see Figure 1(b). Planar orthogonal drawings of trees on a fixed set of points in the plane have been explored recently, see e.g., [4, 10, 15]; in these drawings every edge is drawn as an orthogonal path between two points, and the edges are non-intersecting. A *planar L-shaped drawing* is a simple type of planar orthogonal drawing in which every edge is an orthogonal path of exactly two segments. Such a path is called an *L-shaped edge*. For example see the top-view caterpillar in Figure 1(b) together with a planar L-shaped drawing on a given point set. Biedl et al. [4] proved that every top-view caterpillar on n vertices has a planar L-shaped drawing on every set of $O(n \log n)$ points in the plane that is in *general orthogonal position*, meaning that no two points have the same x - or y -coordinate.

1.1 Our Contributions

In Section 2 we study rollercoasters and prove Conjecture 1. In fact we prove something stronger: every sequence of n distinct numbers contains two rollercoasters of total length n . Our proof is constructive and yields a linear-time algorithm for computing such rollercoasters. We also extend our result to rollercoasters whose runs are of length at least k , for $k > 3$. Then we present an $O(n \log n)$ -time algorithm for computing a longest rollercoaster, extending

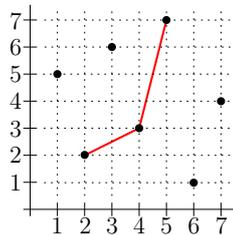
the classical algorithm for computing a longest increasing subsequence. This algorithm can be implemented in $O(n \log \log n)$ time if each number in the input sequence is an integer that fits in a constant number of memory words. Then we give an estimate on the number of permutations of $\{1, \dots, n\}$ that are rollercoasters. In Section 3 we prove, by using Conjecture 1, that every n -vertex top-view caterpillar has a planar L-shaped drawing on every set of $\frac{25}{3}(n+4)$ points in the plane in general orthogonal position.

2 Rollercoasters

In this section we investigate lower bounds for the length of a longest rollercoaster in a sequence of numbers. We also study algorithmic aspects of computing such rollercoasters. First we prove Conjecture 1: any sequence of n distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$. Observe that the length 4 sequence $(3, 4, 1, 2)$ has no rollercoaster, so we will restrict to $n \geq 5$ in the remainder of this section. Also, due to the following proposition we assume that $n \geq 8$.

► **Proposition 1.** *Every sequence of $n \in \{5, 6, 7\}$ distinct real numbers contains a rollercoaster of length at least 3. This bound is tight in the worst case.*

Proof. By applying Theorem 2 with $a = b = 2$ we get that every sequence of at least $ab + 1 = 5$ distinct numbers contains an increasing or a decreasing subsequence of length at least 3. This subsequence is a rollercoaster of length at least 3. For the tightness of this bound, consider the sequence $(5, 2, 6, 3, 7, 1, 4)$, depicted in the figure below. It has length 7 and its longest rollercoaster has length 3.

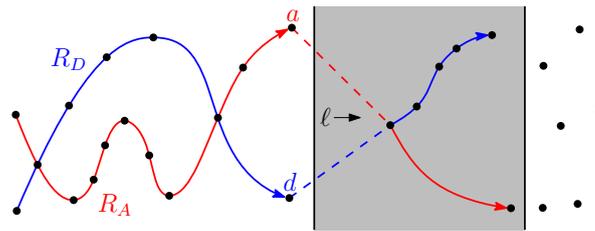


We refer to a polygonal path as a *chain*. We define an *ascent* (resp., a *descent*) as an increasing (resp., a decreasing) sequence. We define a k -*ascent* (resp., a k -*descent*) as an ascent (resp., a descent) with at least k elements. We also use k -*ascent* and k -*descent* to refer to increasing and decreasing chains with at least k points, respectively. With this definition, a rollercoaster is a sequence in which every run is either a 3-*ascent* or a 3-*descent*. We refer to the rightmost run of a rollercoaster as its *last run*.

2.1 A Proof of Conjecture 1

In this section we prove the following theorem, which is a restatement of Conjecture 1. Our proof is constructive, and yields a linear-time algorithm for finding such a rollercoaster.

► **Theorem 4.** *Every sequence of $n \geq 8$ distinct real numbers contains a rollercoaster of length at least $\lceil n/2 \rceil$; such a rollercoaster can be computed in linear time. The lower bound of $\lceil n/2 \rceil$ is tight in the worst case.*



■ **Figure 2** One iteration of algorithm: Constructing two pseudo-rollercoasters.

Consider a sequence with $n \geq 8$ distinct real numbers, and let P be its point-set translation with points p_1, \dots, p_n that are ordered from left to right. We define a *pseudo-rollercoaster* as a sequence in which every run is a 3-ascend or a 3-descent, except possibly the first run. That is, the first run of a pseudo-rollercoaster could be of length at most two, while the other runs are of length at least three. We present an algorithm that computes two pseudo-rollercoasters R_1 and R_2 in P such that $|R_1| + |R_2| \geq n$; the length of the longer one is at least $\lceil n/2 \rceil$. Then with a more involved proof we show how to extend this longer pseudo-rollercoaster to obtain a rollercoaster of length at least $\lceil n/2 \rceil$; this will prove the lower bound.

2.1.1 An Algorithm

First we provide a high-level description of our algorithm as depicted in Figure 2. Our algorithm is iterative, and proceeds by sweeping the plane by a vertical line ℓ from left to right. We maintain the following invariant:

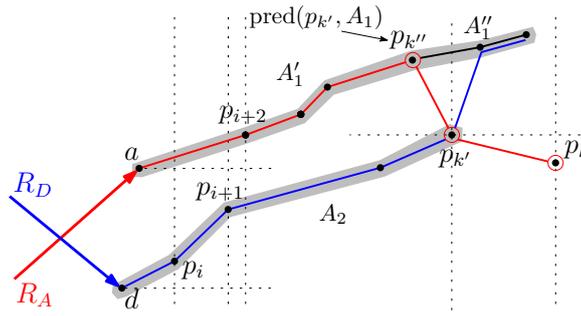
► **Invariant.** *At the beginning of every iteration we have two pseudo-rollercoasters whose union is the set of all points to the left of ℓ and such that the last run of one of them is an ascent and the last run of the other one is a descent. Furthermore, these two last runs have a point in common.*

During every iteration we move ℓ forward and try to extend the current pseudo-rollercoasters. If this is not immediately possible with the next point, then we move ℓ farther and stop as soon as we are able to split all the new points into two chains that can be appended to the current pseudo-rollercoasters to obtain two new pseudo-rollercoasters that satisfy the invariant. See Figure 2.

Now we present our iterative algorithm in detail.

The First Iteration: We take the leftmost point p_1 , and initialize each of the two pseudo-rollercoasters by p_1 alone. We may consider one of the pseudo-rollercoasters to end in an ascent and the other pseudo-rollercoaster to end in a descent. The two runs share p_1 .

An Intermediate Iteration: By the above invariant we have two pseudo-rollercoasters R_A and R_D whose union is the set of all points to the left of ℓ and such that the last run of one of them, say R_A , is an ascent and the last run of R_D is a descent. Furthermore, the last run of R_A and the last run of R_D have a point in common. During the current iteration we make sure that every swept point will be added to R_A or R_D or both. We also make sure that at the end of this iteration the invariant will hold for the next iteration. Let a and d denote the rightmost points of R_A and R_D , respectively; see Figure 2. Observe that a lies above d . Let p_i be the first point to the right of ℓ . If p_i is above a , we add p_i to R_A to complete this iteration. Similarly, if p_i is below d , we add p_i to R_D to complete this iteration. In either



■ **Figure 3** Illustration of an intermediate iteration of the algorithm.

case we get two pseudo-rollercoasters that satisfy the invariant for the next iteration. Thus we may assume that p_i lies below a and above d .

Consider the next point p_{i+1} . (If there is no such point, go to the last iteration; see below.) Suppose by symmetry that p_{i+1} lies above p_i as depicted in Figure 3. Then d, p_i, p_{i+1} forms a 3-ascent. Continue considering points p_{i+2}, \dots, p_k until for the first time, there is a 3-descent in a, p_i, \dots, p_k . In other words, k is the smallest index for which a, p_i, \dots, p_k contains a descending chain of length 3. (If we run out of points before finding a 3-descent, then go to the last iteration.)

Without p_k there is no descending chain of length 3. Thus the longest descending chain has two points, and by Theorem 3, the sequence $P' = a, p_i, p_{i+1}, \dots, p_{k-1}$ is the union of two ascending chains. We give an algorithm to find two such chains A_1 and A_2 with A_1 starting at a and A_2 starting at p_i . The algorithm also finds the 3-descent ending with p_k . For every point $q \in A_2$ we define its A_1 -predecessor to be the rightmost point of A_1 that is to the left of q . We denote the A_1 -predecessor of q by $\text{pred}(q, A_1)$.

The algorithm is as follows: While moving ℓ forward, we denote by r_1 and r_2 the rightmost points of A_1 and A_2 , respectively; at the beginning $r_1 = a$, $r_2 = p_i$, and $\text{pred}(p_i, A_1) = a$. Moreover, we maintain this invariant that $\text{pred}(r_2, A_1)$ is above r_2 . Let p be the next point to be considered. If p is above r_1 then we add p to A_1 . If p is below r_1 and above r_2 , then we add p to A_2 and set $\text{pred}(p, A_1) = r_1$; notice that this assignment satisfies the invariant. If p is below r_2 , then we find our desired first 3-descent formed by (in backwards order) $p_k = p$, $p_{k'} = r_2$, and $p_{k''} = \text{pred}(r_2, A_1)$. See Figure 3. This algorithm runs in time $O(k - i)$, which is proportional to the number of swept points.

We add point d to the start of chain A_2 . The resulting chains A_1 and A_2 are shaded in Figure 3. Observe that A_2 ends at $p_{k'}$. Also, all points of P' that are to the right of $p_{k'}$ (if there are any) belong to A_1 , and lie to the right of $p_{k''}$, and form an ascending chain. Let A_1'' be this ascending chain. Let A_1' be the sub-chain of A_1 up to $p_{k''}$; see Figure 3. Now we form one pseudo-rollercoaster (shown in red) consisting of R_A followed by A_1' and then by the descending chain $p_{k''}, p_{k'}, p_k$. We form another pseudo-rollercoaster (shown in blue) consisting of R_D followed by A_2 and then by A_1'' . We need to verify that the ascending chain added after d has length at least 3. This chain contains d, p_i and $p_{k'}$. This gives a chain of length at least 3 unless $k' = i$, but in this case $p_{k''} = a$, so p_{i+1} is part of A_1' (because p_{i+1} is above p_i) and consequently part of this ascending chain. Thus we have constructed two longer pseudo-rollercoasters whose union is the set of all points up to point p_k , one ending with a 3-ascent and one with a 3-descent and such that the last two runs share the point $p_{k'}$. Figure 4(a) shows an intermediate iteration.

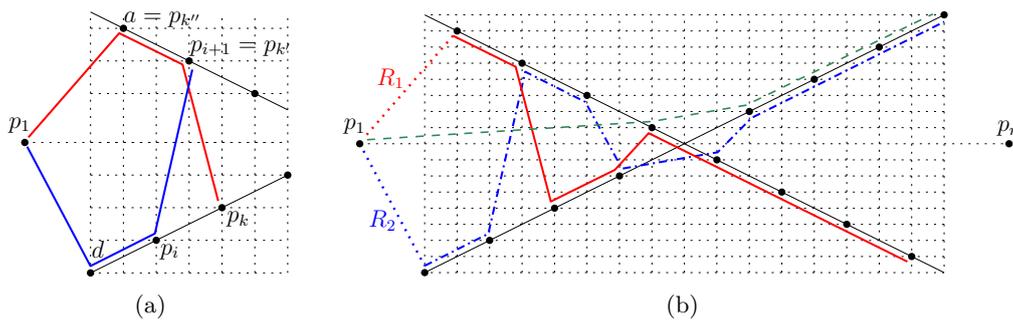


Figure 4 (a) An intermediate iteration. (b) A point set for which any rollercoaster of length at least $n/4 + 3$ does not contain p_1 and p_n . The green (dashed) rollercoaster, which contains p_1 , has length $n/4 + 2$. The red (solid) and blue (dash-dotted) chains are the two rollercoasters returned by our algorithm.

The Last Iteration: If there are no points left, then we terminate the algorithm. Otherwise, let p_i be the first point to the right of ℓ . Let a and d be the endpoints of the two pseudo-rollercoasters obtained so far, such that a is the endpoint of an ascent and d is the endpoint of a descent. Notice that p_i is below a and above d , because otherwise this iteration would be an intermediate one. Moreover, the remaining points p_i, \dots, p_n do not contain a 3-ascent together with a 3-descent, again, because otherwise this iteration would be an intermediate one. If p_i is the last point, i.e., $i = n$, then we discard this point and terminate this iteration. Assume that $i \neq n$, and suppose by symmetry that the next point p_{i+1} lies above p_i . In this setting, by Theorem 3 and as described in an intermediate iteration, with the remaining points, we can get two ascending chains A_1 and A_2 such that A_2 contains at least two points. By connecting A_1 to a and A_2 to d we get two pseudo-rollercoasters whose union is all the points (in this iteration we do not need to maintain the invariant).

Final Refinement: At the end of algorithm, we obtain two pseudo-rollercoasters R_1 and R_2 that share p_1 and such that their union contains all points of P , except possibly p_n . Thus, $|R_1| + |R_2| \geq n$, and the length of the longer one is at least $\lceil \frac{n}{2} \rceil$.

Recall that every run of pseudo-rollercoasters R_1 and R_2 is a 3-ascent or a 3-descent, except possibly the first run. If the first run of R_1 (resp., R_2) contains only two points, then we remove p_1 to obtain a rollercoaster \mathcal{R}_1 (resp., \mathcal{R}_2). Therefore, we obtain two rollercoasters whose union contains all points, except possibly p_1 and p_n .

This is the end of our algorithm. In the next section we analyze the length of the resulting rollercoaster, the tightness of the claimed lower bound, and the running time of the algorithm.

2.1.2 Length and Running-Time Analysis

Our algorithm computes two rollercoasters \mathcal{R}_1 and \mathcal{R}_2 consisting of all points of P , except possibly p_1 and p_n . Thus, the total length of these rollercoasters is at least $n - 2$, and the length of the longer one is at least $\lceil \frac{n-2}{2} \rceil$. In the full version of the paper (see [3]) we show how to improve this bound to $\lceil \frac{n}{2} \rceil$ by revisiting the first and last iterations of our algorithm with some case analysis.

We note that there are point sets, with n points, for which every rollercoaster of length at least $n/4 + 3$ does not contain any of p_1 and p_n ; see e.g., the point set in Figure 4(b). To verify the tightness of the $\lceil n/2 \rceil$ lower bound, consider a set of n points in the plane where

$\lfloor n/2 \rfloor$ of the points lie on a positive-slope line segment in the $(-, +)$ -quadrant and the other $\lfloor n/2 \rfloor$ points lie on a positive-slope line segment in the $(+, -)$ -quadrant.

To verify the running time, notice that the first iteration and final refinement take constant time, and the last iteration is essentially similar to an intermediate iteration. As described in an intermediate iteration the time complexity to find a 3-ascent and a 3-descent for the first time, together with the time complexity to compute chains A'_1 , A''_1 , and A_2 is $O(k - i)$, which is linear in the number of swept points p_i, \dots, p_k . Based on this and the fact that every point is considered only in one iteration, our algorithm runs in $O(n)$ time.

2.2 An Extension

In this section we extend our result to k -rollercoasters. A k -rollercoaster is a sequence of real numbers in which every run is either a k -ascent or a k -descent.

► **Theorem 5.** *Let $k \geq 4$ be an integer. Then every sequence of $n \geq (k - 1)^2 + 1$ distinct real numbers contains a k -rollercoaster of length at least $\frac{n}{2(k-1)} - \frac{3k}{2}$. Moreover, for every $n > 0$ there exists a sequence of n distinct real numbers whose longest k -rollercoaster has length at most $\lceil \frac{n}{k-1} \rceil$.*

Proof. Our proof of the lower bound follows the same iterative approach of the proof of Theorem 4. Consider a sequence of n distinct real numbers and its point-set translation p_1, \dots, p_n . We sweep the plane by a line ℓ , and maintain two k -rollercoasters R_A and R_D to the left of ℓ such that the last run of R_A is an ascent and the last run of R_D is a descent. In each iteration, except the last one, we move ℓ forward and stop as soon as we see a k -ascent A and a k -descent D in the swept points. Then we attach D to R_A , and A to R_D . To achieve the claimed lower bound, we make sure that the total length of A and D is at least $1/(k - 1)$ times the number of swept points.

Consider an intermediate iteration where p_i lies below the rightmost point of R_A and above the rightmost point of R_D . Let m be the number of swept points in this iteration and let $P' = (p_i, p_{i+1}, \dots, p_{i+m-2}, p_{i+m-1})$ be the sequence of these points. Notice that $m \geq 2k - 1$ because we need to sweep at least $2k - 1$ points to get a k -ascent and a k -descent, which may share one point. Our strategy for stopping ℓ ensures that P' contains a k -ascent and a k -descent, while $P'' = (p_i, \dots, p_{i+m-2})$ may contain only one of them but not both. Without loss of generality assume that P'' does not contain a k -descent. Since $m - 1 \geq 2k - 2$, there exists an integer $\alpha \geq 2$ for which

$$(\alpha - 1)(k - 1) < m - 1 \leq \alpha(k - 1). \tag{1}$$

The left-hand side of Inequality (1) implies that P'' has at least $(\alpha - 1)(k - 1) + 1$ points. Having this and our assumption that P'' does not contain a k -descent, Theorem 2 implies that P'' contains an increasing subsequence of length at least α . We take the longest increasing and the longest decreasing subsequences in P' as A and D , respectively. Observe that $|A| \geq \max\{k, \alpha\}$ and $|D| = k$. This and the right-hand side of Inequality (1) imply that

$$|A| + |D| \geq \alpha + k \geq \frac{m - 1}{k - 1} + k > \frac{m}{k - 1},$$

which means that the total length of A and D is at least $1/(k - 1)$ times the number of swept points. In the last iteration if we sweep at most $(k - 1)^2$ points then we discard all of them. But if we sweep $m > (k - 1)^2$ points then by an argument similar to the one above there exists an integer α , with $\alpha \geq m/(k - 1)$, for which we get either an α -ascent or an α -descent, which contains at least $1/(k - 1)$ fraction of the swept points.

The first iteration is similar to the one in the proof of Theorem 4: we assume the existence of an ascent and a descent that end at the first point. At the end of algorithm if the first run of any of R_A and R_D contains k' points, for some $k' < k$, then by removing $k' - 1 (\leq k - 2)$ points from its first run we get a valid k -rollercoaster. The total length of the resulting two k -rollercoasters is

$$|R_A| + |R_D| \geq \frac{n - (k - 1)^2}{k - 1} - 2(k - 2),$$

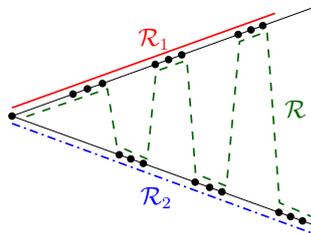
where the length of the longer one is at least

$$\frac{n - (k - 1)^2}{2(k - 1)} - (k - 2) > \frac{n}{2(k - 1)} - \frac{3(k - 1)}{2}.$$

This finishes our proof of the lower bound. To verify the upper bound, consider a set of n points that are placed in the main-diagonal cells of a $(k - 1) \times (k - 1)$ grid, such that every cell contains at most $\lceil \frac{n}{k-1} \rceil$ points that are placed on a positive-slope line. ◀

2.3 Algorithms for a Longest Rollercoaster

In this section we study algorithmic aspects of computing a longest rollercoaster in a given sequence S of n distinct real numbers. By Theorem 4 we can compute a rollercoaster of length at least $\lceil n/2 \rceil$ in $O(n)$ time. However this rollercoaster may not necessarily be a longest one. If we run our algorithm of Section 2.1.1 on the point set in the figure below, then it returns two rollercoasters \mathcal{R}_1 and \mathcal{R}_2 each of length at most $\lceil \frac{n}{2} \rceil$ while the longest rollercoaster \mathcal{R} has length n . In this section, first we adapt the existing $O(n \log n)$ -time algorithm for computing a longest increasing subsequence in S to compute a longest rollercoaster in S within the same time bound. Then we show that if S is a permutation of $\{1, \dots, n\}$, then we can compute a longest rollercoaster in $O(n \log \log n)$ time.



First we recall Fredman’s version of the $O(n \log n)$ -time algorithm for computing a longest increasing subsequence [9]; for more information about longest increasing subsequence, see Romik [14]. We maintain an array $R[i]$, which initially has $R[1] = S[1]$ and is empty otherwise. Then as i proceeds from 2 to n , we find the largest l for which $R[l] < S[i]$, and set $R[l + 1] = S[i]$; if $S[i]$ is smaller than all elements of R , then $l = 0$. This insertion ensures that every element $R[l]$ stores the smallest element of $S[1..i]$ in which an increasing subsequence of length l ends. After all elements of S have been processed, the index of the last non-empty element of R is the largest length of an increasing sequence; the corresponding sequence can also be retrieved from R . Notice that R is always sorted during the above process. So, the proper location of $S[i]$ in R can be computed in $O(\log n)$ time by a predecessor search, which can be implemented as a binary search. Therefore, this algorithm runs in $O(n \log n)$ time.

To compute a longest rollercoaster we need to extend this approach. We maintain six arrays $R(w, h)$ with $w \in \{\text{inc}, \text{dec}\}$ and $h \in \{2, 3_+, 3'_+\}$ where inc stands for “increasing”, dec stands for “decreasing”, and both 3_+ and $3'_+$ stand for any integer that is at least 3 (we will see the difference between 3_+ and $3'_+$ later when we fill the arrays). We define a

18:10 Rollercoasters and Caterpillars

w - h -rollercoaster to be a rollercoaster whose last run has h points and is increasing if $w = \text{inc}$ and decreasing if $w = \text{dec}$. We insert $S[i]$ into arrays $R(\text{inc}, h)$ such that after this insertion the following invariants hold:

- The array $R(\text{inc}, 2)[l]$ stores the smallest element of $S[1..i]$ in which an inc-2-rollercoaster of length l ends. The array $R(\text{dec}, 2)[l]$ stores the largest element of $S[1..i]$ in which a dec-2-rollercoaster of length l ends.
- The array $R(\text{inc}, 3_+)[l]$ stores the smallest element of $S[1..i]$ in which an inc-3₊-rollercoaster of length l ends. The array $R(\text{dec}, 3_+)[l]$ stores the largest element of $S[1..i]$ in which an dec-3₊-rollercoaster of length l ends.
- The array $R(\text{inc}, 3'_+)[l]$ stores the largest element of $S[1..i]$ in which an inc-3₊-rollercoaster of length l ends. The array $R(\text{dec}, 3'_+)[l]$ stores the smallest element of $S[1..i]$ in which a dec-3₊-rollercoaster of length l ends. These arrays will be used when the last run of the current rollercoaster changes from an ascent to a descent, and vice versa.

We insert $S[i]$ into arrays $R(\text{dec}, h)$ so to maintain analogous aforementioned invariants. To achieve these invariants we insert $S[i]$ as follows:

- $R(\text{inc}, 2)$: Find the largest index l such that $R(\text{dec}, 3'_+)[l] < S[i]$. If $S[i] < R(\text{inc}, 2)[l+1]$ then update $R(\text{inc}, 2)[l+1] = S[i]$.
- $R(\text{inc}, 3_+)$: Find the largest indices l_1 and l_2 such that $R(\text{inc}, 2)[l_1] < S[i]$ and $R(\text{inc}, 3_+)[l_2] < S[i]$. Let $l = \max\{l_1, l_2\}$. If $S[i] < R(\text{inc}, 3_+)[l+1]$ then update $R(\text{inc}, 3_+)[l+1] = S[i]$.
- $R(\text{inc}, 3'_+)$: Find the largest index l_1 and l_2 such that $R(\text{inc}, 2)[l_1] < S[i]$ and $R(\text{inc}, 3'_+)[l_2] < S[i]$. Let $l = \max\{l_1, l_2\}$. If $S[i] > R(\text{inc}, 3'_+)[l+1]$ then update $R(\text{inc}, 3'_+)[l+1] = S[i]$.
- The arrays $R(\text{dec}, h)$ are updated in a similar fashion.

Since our arrays $R(w, h)$ are not necessarily sorted, we cannot perform a predecessor search to find proper locations of $S[i]$. To insert $S[i]$ we need to *find* the largest index l such that $R(w, h)[l]$ is smaller (or, alternatively, larger) than $S[i]$ for some w and h , and also need to *update* contents of these arrays. Thereby, if A is an $R(w, h)$ array, we need to perform the following two operations on A :

- FindMax($A, S[i]$): Find the largest index l such that $A[l] > S[i]$ (or $A[l] < S[i]$).
- Update($A, l, S[i]$): Set $A[l] = S[i]$.

We implement each $R(w, h)$ as a Fenwick tree [8], which supports FindMax and Update in $O(\log n)$ time. Thus, the total running time of our algorithm is $O(n \log n)$. After all elements of S have been processed, the largest length of a rollercoaster is the largest value l for which $R(w, 3_+)[l]$ or $R(w, 3'_+)[l]$ is not empty; the corresponding rollercoaster can also be retrieved from arrays $R(w, h)$, by keeping the history of the way the elements of these arrays were computed, and then rolling back the computation.

A Longest Rollercoaster in Permutations: Here we consider a special case where our input sequence S consists of n distinct integers, each of which can be represented using at most c memory words for some constant $c \geq 1$, in a RAM model with logarithmic word size. In linear time, we can sort S , using Radix Sort, and then hash it to a permutation of $\{1, \dots, n\}$. This reduces the problem to finding a longest rollercoaster in a permutation of $\{1, \dots, n\}$. The longest increasing subsequence of such a sequence can be computed in $O(n \log \log n)$ time by using a van Emde Boas tree [18], which supports predecessor search and updates in $O(\log \log n)$ time.⁸ To compute a longest rollercoaster in the same time, we need a data

⁸ We note that a longest increasing subsequence of a permutation can also be computed in $O(n \log \log k)$ time (see [5]) where k is the largest length of an increasing sequence. However, in our case, the largest

structures that supports FindMax and Update in permutations in $O(\log \log n)$ time. In the full version of the paper (see [3]) we show how to obtain such a data structure by using van Emde Boas trees combined with some other structures.

► **Lemma 6.** *Let A be an array with n elements from the set $\{0, 1, \dots, n\}$ such that each non-zero number occurs at most once in A . We can construct, in linear time, a data structure that performs FindMax and Update operations in $O(\log \log n)$ amortized time.*

With Lemma 6 in hand, we can compute a longest rollercoaster in S in $O(n \log \log n)$ time. We note that this algorithm can also compute a longest increasing subsequence by maintaining only the array $R(\text{inc}, 3_+)$.

Notice that both of our algorithms (for general sequences and for permutations) can be generalized to compute a longest k -rollercoaster in $O(kn \log n)$ time and in $O(kn \log \log n)$ time, respectively. A straightforward way is to maintain $2k$ arrays $R(w, h)$ with $w \in \{\text{inc}, \text{dec}\}$ and $h \in \{2, \dots, k-1, k_+, k'_+\}$ and fill them in a way analogous to what we did for rollercoasters. The following theorem summarizes our results in this section.

► **Theorem 7.** *Let $k \geq 3$ be an integer. Then a longest k -rollercoaster in every sequence of n distinct real numbers can be computed in $O(kn \log n)$ time, and a longest k -rollercoaster in every permutation of $\{1, \dots, n\}$ can be computed in $O(kn \log \log n)$ time.*

2.4 Counting Rollercoaster Permutations

In this section we estimate the number $r(n)$ of permutations of $\{1, 2, \dots, n\}$ that are rollercoasters. A brief table follows:

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14
$r(n)$	1	0	2	2	14	42	244	1208	7930	52710	40580	3310702	29742388	285103536

This is sequence [A277556](#) in the *On-Line Encyclopedia of Integer Sequences* [16].

The first step is to rephrase the condition that a permutation is a rollercoaster in the language of ascents and descents. Given a length- n permutation $\pi = \pi_1 \pi_2 \cdots \pi_n$, its *descent word* $u(\pi)$ is defined to be $u_1 u_2 \cdots u_{n-1}$ where $u_i = \mathbf{a}$ if $\pi_i < \pi_{i+1}$ and \mathbf{b} otherwise. For example if $\pi = 2, 4, 6, 1, 3, 5$, then $u(\pi) = \mathbf{aaba}$. Notice that π is a rollercoaster if and only if every maximal contiguous subsequence of $u(\pi)$, that consists of only \mathbf{a} 's or \mathbf{b} 's, has length at least two. In other words, π is a rollercoaster if and only if $u(\pi)$ does not have an isolated \mathbf{a} or an isolated \mathbf{b} ; in fact $u(\pi)$ does not contain patterns $\{\mathbf{aba}, \mathbf{bab}\}$, and also begins and ends with either \mathbf{aa} or \mathbf{bb} . The set of all such descent words $u(\pi)$ is given by the expression

$$(\mathbf{aaa}^* + \mathbf{bbb}^*)^*.$$

This expression specifies that every increasing run and every decreasing run must contain at least three elements. Since this description is a regular expression, one can, in principle, obtain the asymptotic behavior of $r(n)$ using the techniques of [2], but the calculations appear to be formidable.

Instead, we follow the approach of Ehrenborg and Jung [6]. This is based on specifying sets of permutations through pattern avoidance. We say a word w *avoids* a set of words S if no contiguous subword of w belongs to S . Although rollercoasters are not specifiable in terms of a finite set of avoidable patterns, they “almost are”. Consider the patterns

length of a rollercoaster is $\Omega(n)$.

$\{\mathbf{aba}, \mathbf{bab}\}$. Every descent word of a rollercoaster must avoid both these patterns, and every word avoiding these patterns that *also* begins and ends with either \mathbf{aa} or \mathbf{bb} is the descent word of some rollercoaster. Let $s(n)$ be the number of permutations of length n whose descent word avoids $\{\mathbf{aba}, \mathbf{bab}\}$. Then $r(n) = \Theta(s(n))$. From [6, Prop. 5.2] we know that $s(n) \sim c \cdot n! \cdot \lambda^{n-3}$ where $\lambda \doteq 0.6869765032 \dots$ is the root of a certain equation. It follows that $r(n) \sim c' \cdot n! \cdot \lambda^{n-3}$ where c' is a constant, approximately 0.204.

3 Caterpillars

In this section we study the problem of drawing a top-view caterpillar, with L-shaped edges, on a set of points in the plane that is in general orthogonal position. Recall that a top-view caterpillar is an ordered caterpillar of degree 4 such that the two leaves adjacent to each vertex lie on opposite sides of the spine; see Figure 1(b) for an example. The best known upper bound on the number of required points for a planar L-shaped drawing of the n -vertex top-view caterpillar is $O(n \log n)$, for all n ; this bound is due to Biedl et al. [4]. We use Theorem 4 and improve this bound to $\frac{25}{3}n + O(1)$.

In every planar L-shaped drawing of a top-view caterpillar, every node of the spine, except for the two endpoints, must have its two incident spine edges aligned either horizontally or vertically. Such a drawing of the spine (which is essentially a path) is called a *straight-through* drawing. It has been proved in [4] that every n -vertex path has an x -monotone straight-through drawing on any set of at least $c \cdot n \log n$ points, for some constant c . The following theorem improves this bound.

► **Theorem 8.** *Any path of n vertices has an x -monotone straight-through drawing on any set of at least $3n-3$ points in the plane that is in general orthogonal position.*

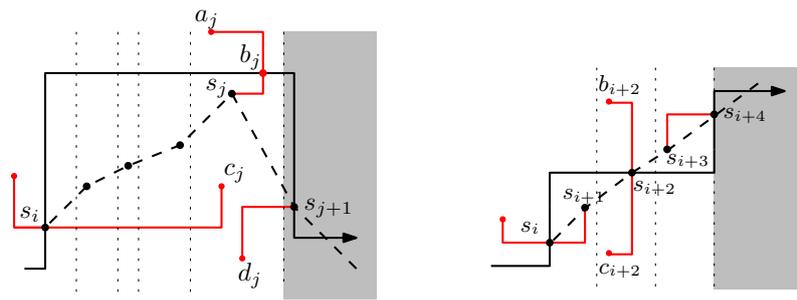
Proof. Fix an arbitrary set of $3n-3$ points. As in the proof of Theorem 4, find two pseudo-rollercoasters that together cover all but the last point and that both contain the first point. Append the last point to both sets; we hence obtain two subsequences R_1, R_2 with $|R_1| + |R_2| \geq 3n-1$ and for which all but the first and last run have length at least 3.

We may assume $|R_1| \geq \frac{3}{2}n - \frac{1}{2}$, and will find the straight-through drawing within it. To do so, consider any run r of R_1 that is neither the first nor the last run, and that has even length (hence length at least 4). By removing from r one point that is not shared with an adjacent run, we turn it into a run of odd length. Let R' be the subsequence that results after applying this to every such run of R_1 ; then R' satisfies that every run except the first and last one has odd length. Observe that we can find an x -monotone straight-through drawing of length $|R'|$ on this, see e.g. the black path in Figure 5 that is drawn on the black points.

It remains to argue that $|R'| \geq n$. Let r_1, \dots, r_ℓ be the runs of R_1 , and assign to each run r_i all but the last point of r_i (the last point of r_i is counted with r_{i+1} , or not counted at all if $i = \ell$). Therefore $|r_1| + \dots + |r_\ell| = |R_1| - 1 \geq \frac{3}{2}n - \frac{3}{2}$. For each r_i with $2 \leq i \leq \ell-1$, we remove a point only if $|r_i| \geq 3$, hence we keep at least $\frac{2}{3}|r_i|$ points. Therefore $|R'| \geq |r_1| + |r_\ell| + \sum_{2 \leq i \leq \ell-1} \frac{2}{3}|r_i| = \sum_{1 \leq i \leq \ell} \frac{2}{3}|r_i| + \frac{1}{3}(|r_1| + |r_\ell|) \geq \frac{2}{3} \cdot (\frac{3}{2}n - \frac{3}{2}) + \frac{1}{3}(2+1) = n$ as desired. ◀

To draw top-view caterpillars, we essentially use Theorem 8 and place the spine on the resulting straight-through x -monotone path. But we will get a slightly better factor if we analyze the number of points directly.

► **Theorem 9.** *For every n , the n -vertex top-view caterpillar has a planar L-shaped drawing on any set of $\frac{25}{3}(n+4)$ points in the plane that is in general orthogonal position. Such a*



■ **Figure 6** Placing the next two spine-vertices. (a) $j \leq i + 4$. (b) $j > i + 4$. The dashed line indicates R , the solid line is the spine.

Case 2: $j > i + 4$. See Figure 6(b). We ignore the reserved points corresponding to s_{i+1} and s_{i+3} . We place v_{2k+1} at s_{i+2} and v_{2k+2} at s_{i+4} . Note that by case-assumption s_{i+4} is *not* the end of the run, so this satisfies the invariant. We connect, as leaves, s_{i+1} to v_{2k} (at s_i), and s_{i+3} to v_{2k+2} (at s_{i+4}). The two leaves of v_{2k+1} can be placed in the 5-set of s_{i+2} . We have used four 5-sets and placed two spine-vertices, and have therefore used half of the parsed 5-sets.

This is the end of one iteration. In every iteration, we have used at least $\frac{2}{5}$ th of the parsed 5-sets. Since there were $\frac{5}{6}(n+4)$ 5-sets, we hence can place $\frac{1}{3}(n+4)$ spine-vertices. Since the spine of the n -vertex top-view caterpillar has $\frac{1}{3}(n+4)$ vertices, our claim about the size of the input point set follows. If the input points are given in sorted order, we can find the rollercoaster in linear time, and then we do one scan of the points to find a planar L -shaped drawing. Thus, our claim about the running time follows. ◀

References

- 1 Tanbir Ahmed and Hunter Snevily. Some properties of roller coaster permutations. *Bulletin of the Institute of Combinatorics and its Applications*, 68:55–69, 2013.
- 2 Nicolas Basset. Counting and generating permutations in regular classes. *Algorithmica*, 76(4):989–1034, 2016.
- 3 Therese C. Biedl, Ahmad Biniaz, Robert Cummings, Anna Lubiw, Florin Manea, Dirk Nowotka, and Jeffrey Shallit. Rollercoasters and caterpillars. *CoRR*, abs/1801.08565, 2018. arXiv:1801.08565.
- 4 Therese C. Biedl, Timothy M. Chan, Martin Derka, Kshitij Jain, and Anna Lubiw. Improved bounds for drawing trees on fixed points with L-shaped edges. In *Proceedings of the 25th International Symposium on Graph Drawing and Network Visualization (GD)*, volume 10692 of *Lecture Notes in Computer Science*, pages 305–317. Springer, 2017. Full version in arXiv:1709.01456.
- 5 Maxime Crochemore and Ely Porat. Fast computation of a longest increasing subsequence and application. *Information and Computation*, 208(9):1054–1059, 2010.
- 6 R. Ehrenborg and J. Jung. Descent pattern avoidance. *Advances in Applied Mathematics*, 49:375–390, 2012.
- 7 Paul Erdős and George Szekeres. A combinatorial problem in geometry. *Compositio Mathematica*, 2:463–470, 1935.
- 8 Peter M. Fenwick. A new data structure for cumulative frequency tables. *Software—Practice and Experience*, 24(3):327–336, 1994.
- 9 Michael L. Fredman. On computing the length of longest increasing subsequences. *Discrete Mathematics*, 11(1):29–35, 1975.

- 10 Emilio Di Giacomo, Fabrizio Frati, Radoslav Fulek, Luca Grilli, and Marcus Krug. Orthogeodesic point-set embedding of trees. *Computational Geometry: Theory and Applications*, 46(8):929–944, 2013.
- 11 John M. Hammersley. A few seedlings of research. In *Proceedings of the 6th Berkeley Symposium on Mathematical Statistics and Probability*, pages 345–394. University of California Press, 1972.
- 12 Sergey Kitaev. *Patterns in Permutations and Words*. Springer, 2011.
- 13 S. Linton, N. Ruškuc, and V. Vatter, editors. *Permutation Patterns*. London Mathematical Society Lecture Note Series, vol. 376, Cambridge, 2010.
- 14 Dan Romik. *The Surprising Mathematics of Longest Increasing Subsequences*. Cambridge, 2015.
- 15 Manfred Scheucher. Orthogeodesic point set embeddings of outerplanar graphs. Master’s thesis, Graz University of Technology, 2015.
- 16 N. J. A. Sloane et al. The On-Line Encyclopedia of Integer Sequences. URL: <https://oeis.org>.
- 17 J. Michael Steele. Variations on the monotone subsequence theme of Erdős and Szekeres. In David Aldous, Persi Diaconis, Joel Spencer, and J. Michael Steele, editors, *Discrete Probability and Algorithms*, pages 111–131. Springer New York, 1995.
- 18 Peter van Emde Boas. Preserving order in a forest in less than logarithmic time and linear space. *Information Processing Letters*, 6(3):80–82, 1977.