Array Range Queries

Matthew Skala

University of Manitoba, Winnipeg, Canada
mskala@cs.umanitoba.ca

Abstract. Array range queries are of current interest in the field of data structures. Given an array of numbers or arbitrary elements, the general array range query problem is to build a data structure that can efficiently answer queries of a given type stated in terms of an interval of the indices. The specific query type might be for the minimum element in the range, the most frequently occurring element, or any of many other possibilities. In addition to being interesting in themselves, array range queries have connections to computational geometry, compressed and succinct data structures, and other areas of computer science. We survey recent and relevant past work on this class of problems.

Keywords: array, range query, document retrieval, range search, selection, range frequency, top-k.

1 Introduction

Given an array of numbers or arbitrary elements, the general array range query problem is to build a data structure that can efficiently answer queries of a given type stated in terms of an interval of the indices. For instance, we might ask for the minimum element value that occurs in the range. It is possible to define a nearly endless variety of such problems by making different choices for the type of query to answer, the amount of space allowed, the model of computation, and so on. Much work has been done on these problems, especially in the last few years. Besides the steady procession of improvements in time and space bounds, work on array ranges has led to new techniques and insights in data structure design applicable to many other problems.

In this paper we survey current results on array range queries. In this introduction we first define the scope of the survey and discuss classification of range queries. Then we introduce some commonly-used techniques for these problems, and in subsequent sections we describe current work in the field organized by our classification.

We are primarily interested in range queries on arrays as such; that is, sequences of data indexed by consecutive integers. Geometric problems are typically approached using different techniques and are not the main focus here. The 1999 survey of Agarwal and Erickson gives detailed coverage of geometric range query work up to that date [1], and we mention some recent geometric work relevant to array range queries. Similarly, although many of the data structures used

A. Brodnik et al. (Eds.): Munro Festschrift, LNCS 8066, pp. 333–350, 2013.

for array range queries are related to those used for string searching problems, our focus here is not on string search. Array ranges can be naturally generalized to queries on paths in trees, and we discuss some of that work without attempting to cover it all.

Array range queries tend to be distinguished by small input and output sizes. A query typically consists of just the starting and ending indices of the range; in some problems there may also be something else in the query, such as a color or a threshold, but seldom more than a constant number of such things. The answer to a query is usually also of small size, such as a single element or just an answer of "yes" or "no." It is unusual for the output size to be as big as the entire contents of the range.

Unless otherwise specified, we assume a RAM model of computation in which the machine can perform operations on words of length $O(\log n)$ in unit time; data structures of $O(n)$ such words; and a static analysis, in which we are allowed to see the entire array in advance and possibly process it in larger working space to build the data structure, after which the element values cannot change. In dynamic versions of array range query problems, the question arises of what is an update. Changing the value of the element at one index, without changing the number of elements nor the values at any other indices, may be the most natural form of dynamic update for an array. However, because of the way they use geometric data structures internally, many dynamic array range data structures also support insertions and deletions, shifting all later elements up or down by one index position, at no additional cost over changing the value of a single element. Unless otherwise specified, we assume that a dynamic update can be an insertion or deletion.

1.1 Classification of Array Range Queries

The unending desire for novel results has led researchers to consider a bewildering assortment of different kinds of array range query problems, and it may not be possible to fit them all into a consistent classification scheme. However, we can discern a few general categories that may be useful in organizing the discussion. Very often, a single paper may describe several different problems, possibly connected by a common data structure that cuts across our classification.

First, an array range query usually considers one of three things about array element values to determine how they can satisfy the query. We discuss each of these in its own section.

- The array elements may be values that have an order, typically real numbers, with the ordering or magnitude of the numbers relevant to the query result. In this case the element values are often called *weights*. These queries are discussed in Section 2.
- The interesting thing about array elements may be only whether each value does or does not occur in the range at least once, without regard for an ordering of the values nor how many times each value occurs. Elements considered this way are often called *colors*. These queries are discussed in Section 3.

– The number of times each value occurs in the range, not only whether it is zero or nonzero, may be important to the query result. Considering element *frequencies* in this way can be said to combine weights (because some values can be ranked above others) and colors (because the values as such are not ordered). These queries are discussed in Section 4.

When considering the ordered element value (weight), a query may meaningfully be stated in terms of the actual value, or its rank among the values that occur in the query range. Similarly, a query based on color could be concerned with an actual color value, or "the first (or k-th) color to occur in the range." And a frequency query could be in terms of a given value of the frequency (either a raw number or a ratio to the query range length) or as a rank among frequencies, such as the mode. In all three categories, then, we can describe range query problems based on raw value and on rank, and that provides a second dimension for our classifiation.

Having defined the property of interest, and whether to consider it as a raw value or as a rank, we can describe different range query problems depending on what information is given as input to the query.

– The values of interest may be completely determined by the problem, so that a query consists of only a range of indices. Such problems can often be seen as special cases of more general problems, of separate interest because they may be easier than the general versions.
– A single value or rank of interest may be specified in the query.
– A threshold may be part of the query, with all values or ranks above or below the threshold being of interest. In some cases, upper and lower thresholds may give rise to significantly different problems. A closely-related category allows an interval (two thresholds) to be given in the query and then matches values within the interval, giving the effect of the intersection of an upper and a lower threshold query, possibly with better bounds.
– A few problems have been described in which the query includes an exact value or a threshold that can be chosen with generality, but the choice is made during preprocessing, and must be the same for all queries.

Finally, there may be several different forms in which a query can return its result, and different kinds of results from what would otherwise be the same problem may require different data structures and have different bounds.

– The query may return a single element weight, color, or frequency matching the criteria, or a rank on one of those properties. These six possibilities correspond to the first two dimensions of our categorization.
– The query may return the index of an element at which the match occurs. Many queries usually thought of as returning values (the previous category) implicitly do it by returning indices where those values occur; but asking for the index is a slightly stronger question, and may be more difficult.
– The query may return a list of all matching values or indices.
– The query may return only a decision result: "yes" or "no" to whether a matching result occurs in the range at all.

1.2 Techniques for Array Range Query

Many range query data structures begin their preprocessing by doing *rank space reduction* [38,22]. Rank space reduction starts with an array $A[1..n]$ of arbitrary elements and creates an array $B[1..n]$ of integers in the range 1 to n, where each element of $B[i]$ represents the rank of $A[i]$ among the set of values that occur in A. A similar array B' is created to translate the ranks back to element values from A. If original values are ever needed, then the ranks from B can be looked up in B'. This reduction means that the rest of the data structure can deal exclusively with integers the size of n; and in particular, it can use the usual RAM techniques to do predecessor queries in $O(\log \log n)$ time.

If a query consists solely of an interval of indices into an array of size n, only $O(n^2)$ distinct queries are possible. Then we could store the answers to all of them in a table that size, and answer queries in constant time just by looking up the answers in the table. Quadratic space is rarely satisfactory, but if we split the input into blocks of size $\Omega(\sqrt{n})$, there are $O(n)$ intervals that start and end on block boundaries and we could afford to store a constant-sized answer for each of them in linear space. Many array range data structures use such a table as the first step in a recursion that will eventually answer arbitrary queries.

It is also common practice to reduce to some other, already-solved array range query problem. Indeed, many of the array range queries we discuss here originally arise as reductions used to solve other problems. The *rank* and *select* queries of the succinct data structures literature [61,60,72,46,79] are often invoked as building blocks for more complicated array range problems. Given a vector of bits, the rank query asks for the number of 1 bits that occur between the start of the vector and a given index; the select query is the reverse, asking for the index of the i-th 1 bit. Rank and select within a range of indices (instead of only counting from the start) are easy to answer by doing, and subtracting out, one additional rank query at the start of the range.

Wavelet trees [48] are also popular building blocks; they generalize succinct bit vectors with rank and select to larger alphabets while providing a few other queries useful in solving array range problems. Navarro provides a general survey [74]. A wavelet tree stores an array (usually called a string in this context) of elements from some alphabet, in a binary structure that divides the alphabet in half at each level. Each node contains a succinct bit vector naming the subtrees responsible for each remaining array elements at that level. The overall space requirement for the basic data structure is $O(n \log \sigma)$ bits where σ is the size of the alphabet, and it can support rank and select on any letter in $O(\log \sigma)$ time; but it can also be used as part of the solution to many other kinds of queries.

2 Weight Queries

When the values stored in the array have meanings directly relevant to the query, they are usually called *weights*. Weights might not be real numbers, but typically have some structure such as a total order, a group, or a semigroup.

2.1 Range Minimum Query

Range Minimum Query (RMQ) on weights equipped with an order is one of the earliest-studied array range query problems. *Range maximum* is equivalent. This problem has the important property that the minimum of a union of two sets must be the minimum of one of the sets; so the query range can be decomposed into a constant number of smaller ranges and the answer found by solving those as subproblems. Highlights of the work on this problem, including the variation for two-dimensional arrays, are shown in Table 1.

Table 1. Selected results on array range minimum query

space (*bits*)	query time	year	ref.	note
$O(n \log n)$	$O(1)$	1984	[51]	LCA
$O(n \log n)$	$O(\log n)$	1984	[38]	2-D
$O(n \log n)$	$O(1)$	1989,1993	[9,10]	
$4n + o(n)$	$O(1)$	2002	[84]	$O(n \log n)$ bits prep. space
array $+ 2n + o(n)$	$O(1)$	2007	[35]	$2n - o(n)$ space lower bound
$O(n \log n)$	$O(1)$	2007	[4]	2-D, $O(n \log^{(k)} n)$ prep. time
$O(n \log n)$	$O(1)$	2009	[26]	cache-oblivious, $O(n/B)$ prep. time
$2n + o(n)$	$O(1)$	2010	[34]	
array $+ O(n)$	$O(1)$	2012	[14]	2-D
$O(n \log n)$	$O(1)$	2013	[29]	simplified

The one-dimensional array RMQ problem can be reduced to *Lowest Common Ancestor* (LCA), and constant-time linear-space solutions for that problem go back at least as far as the work of Harel and Tarjan in 1984 [51]. It is generally taken for granted (whether explicitly stated or not) that data structures for this problem must return the *index* of a minimum element, not only the *weight*. Berkman and Vishkin's work [9,10] is usually cited as the first significant data structure for RMQ. Their main concern was with doing the preprocessing in a parallel model, and with applying the data structure to other problems beyond range queries; but they achieved constant time RMQ with a linear space data structure by reducing from RMQ to LCA and then, by means of *Cartesian trees*, back to RMQ with the constraint that successive elements differ by at most one. Bender and Farach-Colton [8] give a simplified presentation of this data structure. Preprocessing time can be linear with a careful implementation, and subsequent results have also achieved linear preprocessing time.

Sadakane gives a succinct data structure for RMQ, again as part of a solution to LCA [84]. It uses $4n+o(n)$ bits, but it requires asymptotically more than that ($O(n \log n)$ bits) temporarily during preprocessing. Fischer and Huen improve the data structure space bound to $2n + o(n)$ bits with access to the original array [35], and never require more than that even during preprocessing. They also give a lower bound of $2n-o(n)$ bits, and claim an advantage of not using bit vector rank and select (which may make implementation easier). This is usually cited as the current best result for the one-dimensional array RMQ problem in

the basic RAM model. Fischer [34] later improves it to avoid the requirement for access to the original array. In this volume, Durocher [29] gives a new linear-space data structure with constant time query, using simple, practical techniques.

For RMQ on two-dimensional arrays, Gabow, Bentley, and Tarjan [38] give a solution with $O(n \log n)$ space and preprocessing time, and $O(\log n)$ query. Note that we describe the array as $\sqrt{n}$ by $\sqrt{n}$ for a total of n elements overall to make these results more easily comparable to the one-dimensional case. Amir, Fischer, and Lewenstein [4] give a linear-space, constant query time data structure for two-dimensional array RMQ with $O(n \log^{(k)} n)$ preprocessing time, where $\log^{(k)}$ means logarithm iterated any constant number of times.

Demaine, Landau, and Weimann [26] extend RMQ in several directions at once, as well as reviewing some extensions of the problem in more detail. They give an algorithm for one-dimensional arrays with the usual linear space and constant query time as well as optimal $O(n/B)$ preprocessing time in the cache-oblivious model (B is the block size); they solve *Bottleneck Edge Query* (BEQ), which is a generalization to paths in graphs, in linear space, $O(k)$ query time, and $O(n \log^{(k)} n)$ preprocessing time, as well as giving some results on a dynamic version; and they show a combinatorial lower bound that suggests Cartesian trees cannot usefully be applied to the two-dimensional array RMQ problem. Other techniques than Cartesian trees may still be applicable. Brodal, Davoodi, and Rao [14] give a data structure for two-dimensional RMQ in $O(n/c)$ bits (not words; c can be chosen) plus access to the original array, with linear preprocessing and $O(c)$ query time, and they show a matching lower bound.

2.2 Counting and Reporting

Purely counting elements in an array range is trivial, so the term *range counting* usually refers to counting elements subject to some restriction, such as elements whose weight is in a specified interval. Similarly, *range reporting* on arrays usually refers to reporting elements whose weights are within an interval. These definitions make the array range queries equivalent to geometric range queries on a two-dimensional grid, and the best results use geometric range query techniques. Conversely, solutions to geometric versions of these problems often start with rank-space reduction to integer coordinates, so the equivalence is close.

We summarize interesting results on counting, reporting, and emptiness in Table 2. The 1988 data structure of Chazelle [22] is a precursor to the wavelet tree, without the optimization of using succinct bit vectors in the nodes; it achieves $O(\log n)$ time for counting and $O(\log n + k \log^{\epsilon} n)$ for reporting, where k is the number of results, in linear space. By using a wavelet tree, Mäkinen and Navarro [70] achieve $O(\log n)$ time for counting and $O(k \log n)$ for reporting. Bose et al. [11] improve both these query times by a factor of $\log \log n$. Other significant results also involve $\log^{\epsilon} n$ trade-offs: for instance, $O(\log n + k)$ reporting with $O(n \log^{\epsilon} n)$ space [3]. Nekrich [75,76] gives a good survey of previous work up to 2009, as well as a $O(\log n + k \log^{\epsilon} n)$ dynamic reporting data structure (linear space, $O(\log^{3+\epsilon} n)$ updates) and a $O(\log n / \log \log n + k \log^{\epsilon} n)$ static linear-space reporting data structure.

Table 2. Selected results on range counting and reporting

problem	space	query time	year	ref.
counting	$O(n)$	$O(\log n)$	1988	[22]
	$n + o(n)$	$O(\log n)$	2007	[70]
	$n + o(n)$	$O(\log n / \log\log n)$	2009	[11]
	$n + o(n)$	$O((\log n / \log\log n)^2)^*$	2011	[52]
	$O(n)$	$O(\log \sigma)^\dagger$	2011	[55]
	$n + o(n)$	$O(\log \sigma \log n)^\dagger$	2012	[78]
	compressed	$O(\log \sigma / \log\log n)^\dagger$	2012	[56]
reporting	$O(n)$	$O(\log n + k \log^\epsilon n)$	1988	[22]
	$O(n \log^\epsilon n)$	$O(\log n + k)$	2000	[3]
	$n + o(n)$	$O(k \log n)$	2007	[70]
	$n + o(n)$	$O(k \log n / \log\log n)$	2009	[11]
	$O(n)$	$O(\log n + k \log^\epsilon n)^*$	2007,2009	[75,76]
	$O(n)$	$O((1 + k) \log \sigma)^\dagger$	2011	[55]
	$O(n \log\log n)$	$O(\log \sigma + k \log\log \sigma)^\dagger$	2011	[55]
	$n + o(n)$	$O((\log n + k) \log \sigma \log n)^\dagger$	2012	[78]
	compressed	$O((1 + k) \log \sigma / \log\log n)^\dagger$	2012	[56]

*dynamic †on trees

Range counting results are often presented as barely-discussed corrollaries in papers on range reporting, but He and Munro [52] give a succinct dynamic data structure specifically for counting, with $O((\log n / \log\log n)^2)$ time for queries and updates in the worst case of large alphabet size; they state the bounds in a more complicated form taking into account the possibility of small alphabets, and discuss applications to geometric range counting. He, Munro, and Zhou [55] also propose extending counting and reporting queries to paths in trees; they achieve $O((1 + k) \log \sigma)$ reporting time (to report k elements, σ distinct weights in the tree) with linear space, and $O(\log \sigma + k \log\log \sigma)$ time with $O(n \log\log n)$ space. Patil, Shah, and Thankachan [78] give succinct data structures for tree path counting and reporting at a $\log n$ time penalty. Then He, Munro, and Zhou improve their earlier linear-space results to compressed space [56], also improving the query times by $\log\log n$.

2.3 Other Weight Queries

There is a near-trivial folklore solution to *range sum*, or equivalently *range mean*, in a static array: just precompute and store the sums for all ranges beginning at the start of the array, and then subtract the sums for the endpoints to answer an arbitrary range query in constant time. This approach generalizes to arbitrary dimension by applying the principle of inclusion and exclusion on ranges with one corner fixed at the origin, although the constant in the query time is exponential in the number of dimensions. Fredman [37] introduced the dynamic one-dimensional version of the problem (with updates consisting of changing a single element's value) and gave a $O(\log n)$ time, linear-space solution.

A typical modern approach might use any standard balanced tree augmented with subtree sums to allow query, update, insertion, and deletion in $O(\log n)$ time. Matching $\Omega(\log n)$ bounds have been shown in various models of computation [37,85,50].

There are matching upper and lower bounds of $\Theta(n+m\alpha(m,n))$ time to do m array range sum queries *off-line*, where α is the inverse Ackermann function [23]. Brodnik et al. [18] give a constant-time solution using $O\big((n+U^{O(\log n)})\log U\big)$ bits of memory in the *RAMBO* model of computation, where bits can be shared among memory words, summing integers modulo U. The high-dimensional dynamic array case was extensively studied in the database community around the turn of the century [57,44,24,82]. Almost all of these results still apply when "sum" is generalized to arbitrary group operations; some of the related work can be further generalized to arbitrary semigroups, forming a connection to the results on range minimum because minimum is a semigroup operation.

Range minimum was generalized in one direction to semigroups, but a different generalization is to the *range selection* problem of returning a chosen order statistic. The special case of *range median* was the first to be studied: Krizanc, Morin, and Smid [67,68] gave multiple results for it trading off time and space, with $O(n^\epsilon)$ time for linear space on array ranges. They also considered tree paths. Gagie, Puglisi, and Turpin [43] review existing results on range median and then supersede most of them by showing how to use wavelet trees to answer selection, for general order statistics chosen at query time, in $O(\log \sigma)$ time with a linear-space data structure. Subsequent results by Brodal and Jørgensen [17]; Gfeller and Sanders [45]; and then all four of those authors together [16] include queries in $O(\log n/\log\log n)$ time with linear space, a dynamic version with $O((\log n/\log\log n)^2)$ query time in $O(n\log n/\log\log n)$ space, and various results on other models including a cell-probe lower bound of $O(\log n/\log\log n)$ for dynamic range median if updates are $O(\log^{O(1)} n)$. Jørgensen and Larsen [63] give additional lower bounds, notably including $\Omega(n^{1+\Omega(1)})$ space for constant-time queries; and a linear-space static data structure with query time $O((1+\log k)/\log\log n)$ where k is the order statistic desired, optimal by their lower bounds except for small values of k. In compressed space, He, Munro, and Nicholson [53] give a dynamic data structure with query time, in a recently-posted correction to the conference paper [54], $O((\log n/\log\log n)(\log\sigma/\log\log\sigma))$. The earlier-mentioned tree path counting results of He, Munro, and Zhou [55,56] and of Patil, Shah, and Thankachan [78] can also be applied to tree path selection with nearly the same bounds; in the case of the He, Munro, and Zhou results the denominator in the query time changes from $\log\log n$ to $\log\log\sigma$.

Instead of selecting a single order statistic, we could ask for the first k order statistics: a *top-k* problem. Range reporting answers top-k queries if the results are returned in sorted order, preferably with output-sensitive time, and many current range reporting data structures work that way. However, Brodal et al. [15] describe a linear-space data structure for top-k reporting with $O(k)$ output time, and that is better than the lower bound for range reporting. As we discussed, "range reporting" in current use implies that queries include thresholds

on element weights, in effect an extra dimension of the geometric problem, and that is not part of the definition of top-k queries. Nekrich and Navarro [77] give a linear-space data structure for sorted reporting *with* a threshold in $O(k \log^{\epsilon} n)$ time, as well as some other time-space trade-offs.

3 Color Queries

If we consider array elements primarily with respect to distinctness, possibly imposing an order on the distinct values, we can define array range queries based on these *colors*. Color range query problems often arise in the document retrieval literature, and are often presented along the way to solving other problems rather than as independent results.

Gagie et al. [42] have a useful framework for understanding recent work on colored range queries. As they explain, the *colored range listing* problem is fundamental. Given a range, colored range listing returns the index of the first element of each distinct color in the range. Looking at the array elements is easy; the difficulty lies in removing the duplicates to give output-sensitive time. Some of the first output-sensitive results are due to Janardan and Lopez [62], whose linear-space static data structure reports k results in $O(\log n + k)$ time. Gupta, Janardan, and Smid [49] suggest transforming each element $A[i]$ in the original array A into a point (i, j) on the plane using the largest $j < i$ such that $A[j] = A[i]$, and give dynamic results. Muthukrishnan [73] describes an equivalent transformation in terms of an array C. In either form, appropriate range queries on the transformed input give the answers to colored range listing. Muthukrishnan's data structure uses $O(n)$ space and $O(k)$ query time. The subsequent improvements to range minimum queries allow tighter space bounds with or without compromises on the query time, and much subsequent work on colored range listing comes down to applying better RMQ results in this setting. Gagie, Puglisi, and Turpin [43] describe how to use wavelet trees for range selection, the generalization of range minimum, and apply it to colored range listing. Using range selection instead of range minimum eliminates the need to store the suffix array in the document retrieval application they consider. The recent publication of Gagie et al. on compressed document retrieval [41] includes a good survey of colored range problems within their framework.

Merely counting the distinct colors instead of returning them is the *range color counting* problem. Bozanis et al. [13] give a linear-space $O(\log n)$-time data structure for it. Lai, Poon, and Shi [69] apply a result of Gupta et al. [49] to solve the dynamic version with a $\log^2$ penalty: either $O(n \log n)$ space and $O(\log n)$ query, or linear space and $O(\log^2 n)$ query. Gagie and Kärkkäinen [40] reduce it to compressed space with query time logarithmic in the length of the query range, slightly better than $O(\log n)$. They also give some dynamic results, expanded in the journal version by Gagie et al. [41].

If we ask for colors to be listed in order, combining the distinctness of colors with the ordering of weights, we have the *top-k color reporting* problem. Karpinski and Nekrich [66] give a linear-space data structure for this problem

with optimal $O(k)$ query time. This top-k problem defined by an ordering of the element values is different from a top-k problem defined by frequency rank within the range, as considered in the next section. Authors working on such problems use similar or identical terminology for the two.

4 Frequency Queries

When queries concern element frequencies, most combinations of the other parameters in our classification yield interesting and distinct problems: we can ask about raw frequencies or their ranks, provide exact values or thresholds, and return several different kinds of results. High frequencies and low frequencies often necessitate different techniques and sometimes even have different bounds.

4.1 Queries Related to Raw Frequency

The well-known element uniqueness problem, of determining whether any element in an array occurs more than once, has a lower bound of $\Omega(n \log n)$ in the algebraic decision tree model [7]. Determining whether the frequency exceeds k requires $\Theta(n \log(n/k))$ (matching upper and lower bounds) [71,27]. These bounds also apply to the time for *preprocessing plus one query* on the array range versions of the problems. The array range query of whether any element has frequency at least k in the range, k chosen during preprocessing, is trivial to solve with a linear-space data structure and constant-time queries: just store, for each range starting point, the end of the shortest range for which the answer is "yes." Greve et al. [47] describe that data structure. But for $k > 1$, even if chosen during preprocessing, testing the existence of an element with frequency exactly k in the range is more difficult. Greve et al. [47] show matching upper and lower time bounds of $\Theta(\log n / \log \log n)$ with k chosen at query time, assuming a linear-space data structure, the cell probe model for the lower bound, and word RAM for the upper bound.

A closely related but not identical class of problems considers a threshold on the *proportion* of array elements in a range that contain a given value. These problems represent a stepping stone to the frequency-rank problems in the next subsection. Results for these problems are summarized in Table 3.

Given β fixed at preprocessing time, the *range β-majority* query problem is to return an element that occurs (or all such elements) in at least β proportion of the elements of the query range. A geometric data structure of Karpinski and Nekrich [65] can be applied to solve this in $O(n/\beta)$ space and $O((1/\beta)(\log \log n)^2)$ query time. Durocher et al. [30,31] give a solution in $O(n \log(1+1/\beta))$ space and $O(1/\beta)$ query time, and Gagie et al. [39] in $O(n)$ space and $O((1/\beta) \log \log n)$ query time; in recent work, Belazzougui, Gagie, and Navarro improve the query time to optimal $O(1/\beta)$ [6]. Elmasry et al. [32] study a dynamic version of this problem, mostly in a geometric setting with β fixed and updates consisting of point addition and removal; for arrays, their data structure gives $O(n)$ space and $O((1/\beta) \log n / \log \log n)$ query time, with $O((1/\beta) \log^3 n / \log \log n)$ amortized insertion and $O((1/\beta) \log n)$ amortized deletion.

Table 3. Selected results on array range majority and minority

problem	space	query time	year	ref.
	$O(n/\beta)$	$O(1/\beta)$	2008	[65]
	$O(n \log(1 + 1/\beta))$	$O(1/\beta)$	2011,2013	[30,31]
β-majority	$O(n)$	$O((1/\beta) \log \log n)$	2011	[39]
	$O(n)$	$O((1/\beta) \log n / \log \log n)^*$	2011	[32]
	$O(n)$	$O(1/\beta)$	2012	[6]
	$O(n(H + 1))$	$O(1/\alpha)$	2011	[39]
	$O(n \log n)$	$O(1/\alpha)$	2012	[21]
α-majority	$n \log \log \sigma + O(n)$	$O(1/\alpha)$	2012	[6]
	$O(n)$	$O((1/\alpha) \log \log(1/\alpha))$	2012	[6]
	$n + o(n)$	$O((1/\alpha) \log \log \sigma)$	2012	[6]
α-minority	$O(n)$	$O(1/\alpha)$	2012	[21]

*dynamic

When the proportion threshold is α chosen at query time, it is easy to solve range majority at a factor of $\log n$ space penalty by building copies of a slightly modified β-majority data structure for each power of two value of β and then choosing the closest one at query time. Chan et al. describe that technique [21]. Gagie et al. [39] give a compressed range α-majority data structure ($O(n(H+1))$ words where H is entropy, bounded above by $\log \sigma$ where σ is the number of distinct elements) with $O(1/\alpha)$ query time. Belazzougui, Gagie, and Navarro [6] give multiple new results for this problem with query time varying depending on space, as summarized in our table.

Another possibility is to make the frequency threshold depend on the element value. De Berg and Haverkort [25] describe *significant-presence* queries. An element value has a significant presence in a query range if at least a specified fraction of the elements with that value in the entire array occur within the query range: the range covers a fraction of the color class rather than the color class necessarily covering a fraction of the range. The significant-presence range query is to find all the values which have significant presence in the range. The main results of de Berg and Haverkort concern approximate versions of this problem in an arbitrary-dimensional geometric setting, but they solve the exact problem on one-dimensional arrays with the threshold fixed during preprocessing using linear space and $O(\log n + k)$ query time.

Querying for a threshold on frequency in the opposite direction, that is, considering elements that occur at least once in the query range but less than some number of times, requires significantly different techniques. Chan et al. [21] solve range α-minority (α may be chosen at query time) using $O(n)$ space and $O(1/\alpha)$ query time.

4.2 Queries Related to Frequency Rank

Range *mode* (most frequent element) is arguably a more natural and useful query than range majority; indeed, previous results on range majority are often inspired

by or connected with work on the more difficult question of range mode. Range mode and more general frequency rank problems are of great current interest because of applications in document retrieval: in an array of document identifiers corresponding to a suffix array, high-frequency elements in a range identify the documents that contain a given substring many times.

Krizanc et al. [67,68] introduce the range mode problem for arrays, immediately also generalizing it to paths in trees. They give data structures for arrays and trees with $O(n^{2-2\epsilon})$ space and $O(n^{\epsilon} \log n)$ query time, where ϵ can be chosen. Setting it to $1/2$ gives linear space and $O(\sqrt{n} \log n)$ time. They also give a $O(n^2 \log\log n / \log n)$-space, constant-time data structure, slightly beating the obvious quadratic-sized table of all possible answers. In the same work they consider range median; and the connection between median and mode has persisted in later work by others, with results on both problems often appearing simultaneously.

Petersen [80] improves the $O(n^{\epsilon} \log n)$-time result to $O(n^{\epsilon})$ with the same $O(n^{2-2\epsilon})$ space; however, the hidden constant in the space requirement goes to infinity as ϵ approaches $1/2$, so linear space is no longer achievable. He improves the space bound for constant time to $O(n^2/\log n)$; then, with Grabowski [81], to $O(n^2 \log\log n / \log^2 n)$.

Bose et al. [12] consider approximate range mode, where the goal is to return an element with frequency at least β (chosen at preprocessing) times the frequency of the true mode. Their general data structure requires $O(n/(1-\beta))$ space and $O(\log\log_{1/\beta} n)$ time, but for $\beta \in \{1/2, 1/3, 1/4\}$ they give data structures with constant time and $O(n \log n)$, $O(n \log\log n)$, or $O(n)$ space, respectively. They also give results on an approximate version of the range median.

Chan et al. [19,20] have the best current results on linear-space array range mode, including $O(\sqrt{n/\log n})$ time with linear space; actually $O(\sqrt{n/w})$ on a w-sized word RAM, thus $O(\sqrt{n/B})$ in external memory. They give an argument, based on a reduction from boolean matrix multiplication, suggesting that the $\sqrt{n}$ factor in the time may be difficult or impossible to remove with a linear-space data structure; and some results on the dynamic one-dimensional version where updates consist of changing single element values (not insertion or deletion), and some higher-dimensional geometric range mode queries.

Top-k array range frequency is closely related to *top-k document retrieval*, the problem of returning the documents with top k most occurrences of a search substring, or the top k best values of some relevance function related to counting substring occurrences. Top-k document retrieval reduces to top-k array range frequency on the suffix array; however, because the queries are related to substrings, they correspond to internal nodes of the suffix tree, and there are only a linear number of possible query ranges. True top-k array range frequency would allow a quadratic number of distinct query ranges. This distinction allows the data structures for top-k document retrieval to achieve better results than would be possible for top-k array range frequency; and the techniques used, although related, are not directly applicable to the general array range problems. Hon, Shah, and Vitter [59], and Hon, Shah, and Thankachan [58] have some interesting

recent work on the document retrieval problem. On the array version, there is very little work on the exact problem; recall that merely finding the mode is a difficult problem, and the top-k version must be at least as hard. Some of the work of Janardin and Lopez [62] is applicable to special cases of top-k frequency. Gagie et al. [41], as well as giving results on document retrieval, solve an ϵ-approximate version, in $O((n/\epsilon)\log n)$ space (worst case; they state a more precise bound in terms of entropy) and $O(k \log \sigma \log(1/\epsilon))$ query time. Chan et al. [21] use a one-sided version (one end of the query must be index 0, making the problem much easier) with $O(n)$ space and $O(k)$ query time as part of their range majority data structure.

Top-k array range frequency naturally suggests a selection version, of finding the k-th most frequent element in an array range. That is an open problem in the case of general k. We have described the special case of range mode already. Chan et al. [21] introduce the *range least frequent element* problem, giving a linear-time data structure with $O(\sqrt{n})$ query time and an argument from boolean matrix multiplication suggesting that, as with mode, a significantly better query time may not be possible.

5 Conclusion

We have surveyed work in array range queries, an active field of current data structure research. We have also described a classification scheme for array range query problems. Not all imaginable categories in our classification are covered by existing work; the missing ones may suggest open problems of interest.

Acknowledgement. We gratefully acknowledge the suggestions made by an anonymous reviewer.

References

1. Agarwal, P.K., Erickson, J.: Geometric range searching and its relatives. In: Chazelle, B., Goodman, J.E., Pollack, R. (eds.) Advances in Discrete and Computational Geometry. Contemporary Mathematics, vol. 223, pp. 1–56. American Mathematical Society, Providence (1999)
2. Albers, S., Marchetti-Spaccamela, A., Matias, Y., Nikoletseas, S., Thomas, W. (eds.): ICALP 2009, Part I. LNCS, vol. 5555. Springer, Heidelberg (2009)
3. Alstrup, S., Brodal, G.S., Rauhe, T.: New data structures for orthogonal range searching. In: 41st Annual Symposium on Foundations of Computer Science, FOCS 2000, Redondo Beach, California, USA, November 12-14, pp. 198–207 (2000)
4. Amir, A., Fischer, J., Lewenstein, M.: Two-dimensional range minimum queries. In: Ma, B., Zhang, K. (eds.) CPM 2007. LNCS, vol. 4580, pp. 286–294. Springer, Heidelberg (2007)
5. Asano, T., Nakano, S.-I., Okamoto, Y., Watanabe, O. (eds.): ISAAC 2011. LNCS, vol. 7074. Springer, Heidelberg (2011)

6. Belazzougui, D., Gagie, T., Navarro, G.: Better space bounds for parameterized range majority and minority. In: Dehne, F., Solis-Oba, R., Sack, J.-R. (eds.) WADS 2013. LNCS, vol. 8037, pp. 121–132. Springer, Heidelberg (2013)
7. Ben-Or, M.: Lower bounds for algebraic computation trees (preliminary report). In: Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing, Boston, Massachusetts, April 25-27, pp. 80–86 (1983)
8. Bender, M.A., Farach-Colton, M.: The LCA problem revisited. In: Gonnet, G.H., Viola, A. (eds.) LATIN 2000. LNCS, vol. 1776, pp. 88–94. Springer, Heidelberg (2000)
9. Berkman, O., Vishkin, U.: Recursive *-tree parallel data-structure. In: Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science, FOCS 1989, Research Triangle Park, NC, October 30-November 1, pp. 196–202. IEEE Computer Society Press, Los Alamitos (1989)
10. Berkman, O., Vishkin, U.: Recursive star-tree parallel data structure. SIAM J. Comput. 22(2), 221–242 (1993)
11. Bose, P., He, M., Maheshwari, A., Morin, P.: Succinct orthogonal range search structures on a grid with applications to text indexing. In: Dehne, F., Gavrilova, M., Sack, J.-R., Tóth, C.D. (eds.) WADS 2009. LNCS, vol. 5664, pp. 98–109. Springer, Heidelberg (2009)
12. Bose, P., Kranakis, E., Morin, P., Tang, Y.: Approximate range mode and range median queries. In: Diekert, V., Durand, B. (eds.) STACS 2005. LNCS, vol. 3404, pp. 377–388. Springer, Heidelberg (2005)
13. Bozanis, P., Kitsios, N., Makris, C., Tsakalidis, A.: New upper bounds for generalized intersection searching problems. In: Fülöp, Z., Gécseg, F. (eds.) ICALP 1995. LNCS, vol. 944, pp. 464–474. Springer, Heidelberg (1995)
14. Brodal, G.S., Davoodi, P., Rao, S.S.: On space efficient two dimensional range minimum data structures. Algorithmica 63(4), 815–830 (2012)
15. Brodal, G.S., Fagerberg, R., Greve, M., López-Ortiz, A.: Online sorted range reporting. In: [28], pp. 173–182
16. Brodal, G.S., Gfeller, B., Jørgensen, A.G., Sanders, P.: Towards optimal range medians. Theoret. Comput. Sci. 412(24), 2588–2601 (2011)
17. Brodal, G.S., Jørgensen, A.G.: Data structures for range median queries. In: [28], pp. 822–831
18. Brodnik, A., Karlsson, J., Munro, J.I., Nilsson, A.: An O(1) solution to the prefix sum problem on a specialized memory architecture. In: Navarro, G., Bertossi, L.E., Kohayakawa, Y. (eds.) TCS 2006. IFIP, vol. 209, pp. 103–114. Springer, Boston (2006)
19. Chan, T.M., Durocher, S., Larsen, K.G., Morrison, J., Wilkinson, B.T.: Linear-space data structures for range mode query in arrays. In: Dürr, C., Wilke, T. (eds.) 29th International Symposium on Theoretical Aspects of Computer Science, STACS 2012, Paris, France, February 29-March 3. LIPIcs, vol. 14, pp. 290–301. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2012)
20. Chan, T.M., Durocher, S., Larsen, K.G., Morrison, J., Wilkinson, B.T.: Linear-space data structures for range mode query in arrays. Theory Comput. Sys., 1–23 (2013)
21. Chan, T.M., Durocher, S., Skala, M., Wilkinson, B.T.: Linear-space data structures for range minority query in arrays. In: [36], pp. 295–306
22. Chazelle, B.: A functional approach to data structures and its use in multidimensional searching. SIAM J. Comput. 17(3), 427–462 (1988)

23. Chazelle, B., Rosenberg, B.: The complexity of computing partial sums off-line. Internat. J. Comput. Geom. Appl. 1(1), 33–45 (1991)
24. Chun, S.J., Chung, C.W., Lee, J.H., Lee, S.L.: Dynamic update cube for range-sum queries. In: Apers, P.M.G., Atzeni, P., Ceri, S., Paraboschi, S., Ramamohanarao, K., Snodgrass, R.T. (eds.) Proceedings of the Twenty-seventh International Conference on Very Large Data Bases, Roma, Italy, September 11-14, pp. 521–530. Morgan Kaufmann Publishers (2001)
25. de Berg, M., Haverkort, H.J.: Significant-presence range queries in categorical data. In: Dehne, F., Sack, J.-R., Smid, M. (eds.) WADS 2003. LNCS, vol. 2748, pp. 462–473. Springer, Heidelberg (2003)
26. Demaine, E.D., Landau, G.M., Weimann, O.: On Cartesian trees and range minimum queries. In: [2], pp. 341–353
27. Dobkin, D., Munro, J.I.: Determining the mode. Theoret. Comput. Sci. 12(3), 255–263 (1980)
28. Dong, Y., Du, D.-Z., Ibarra, O. (eds.): ISAAC 2009. LNCS, vol. 5878. Springer, Heidelberg (2009)
29. Durocher, S.: A simple linear-space data structure for constant-time range minimum query. In: Brodnik, A., López-Ortiz, A., Raman, V., Viola, A. (eds.) Munro Festschrift 2013. LNCS, vol. 8066, pp. 48–60. Springer, Heidelberg (2013)
30. Durocher, S., He, M., Munro, J.I., Nicholson, P.K., Skala, M.: Range majority in constant time and linear space. In: Aceto, L., Henzinger, M., Sgall, J. (eds.) ICALP 2011, Part I. LNCS, vol. 6755, pp. 244–255. Springer, Heidelberg (2011)
31. Durocher, S., He, M., Munro, J.I., Nicholson, P.K., Skala, M.: Range majority in constant time and linear space. Inform. and Comput. 222, 169–179 (2013)
32. Elmasry, A., He, M., Munro, J.I., Nicholson, P.K.: Dynamic range majority data structures. In: [5], pp. 150–159
33. Eppstein, D. (ed.): Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Mathematics (SODA 2002), January 6-8. ACM Press, New York (2002)
34. Fischer, J.: Optimal succinctness for range minimum queries. In: López-Ortiz, A. (ed.) LATIN 2010. LNCS, vol. 6034, pp. 158–169. Springer, Heidelberg (2010)
35. Fischer, J., Heun, V.: A new succinct representation of RMQ-information and improvements in the enhanced suffix array. In: Chen, B., Paterson, M., Zhang, G. (eds.) ESCAPE 2007. LNCS, vol. 4614, pp. 459–470. Springer, Heidelberg (2007)
36. Fomin, F.V., Kaski, P. (eds.): SWAT 2012. LNCS, vol. 7357. Springer, Heidelberg (2012)
37. Fredman, M.L.: The complexity of maintaining an array and computing its partial sums. J. ACM 29(1), 250–260 (1982)
38. Gabow, H.N., Bentley, J.L., Tarjan, R.E.: Scaling and related techniques for geometry problems. In: Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing, April 30-May 2, pp. 135–143. ACM Press, Washington, DC (1984)
39. Gagie, T., He, M., Munro, J.I., Nicholson, P.K.: Finding frequent elements in compressed 2D arrays and strings. In: Grossi, R., Sebastiani, F., Silvestri, F. (eds.) SPIRE 2011. LNCS, vol. 7024, pp. 295–300. Springer, Heidelberg (2011)
40. Gagie, T., Kärkkäinen, J.: Counting colours in compressed strings. In: Giancarlo, R., Manzini, G. (eds.) CPM 2011. LNCS, vol. 6661, pp. 197–207. Springer, Heidelberg (2011)

41. Gagie, T., Kärkkäinen, J., Navarro, G., Puglisi, S.J.: Colored range queries and document retrieval. Theoret. Comput. Sci. 483, 36–50 (2013)
42. Gagie, T., Navarro, G., Puglisi, S.J.: Colored range queries and document retrieval. In: Chavez, E., Lonardi, S. (eds.) SPIRE 2010. LNCS, vol. 6393, pp. 67–81. Springer, Heidelberg (2010)
43. Gagie, T., Puglisi, S.J., Turpin, A.: Range quantile queries: Another virtue of wavelet trees. In: Karlgren, J., Tarhio, J., Hyyrö, H. (eds.) SPIRE 2009. LNCS, vol. 5721, pp. 1–6. Springer, Heidelberg (2009)
44. Geffner, S., Agrawal, D., Abbadi, A.E., Smith, T.R.: Relative prefix sums: An efficient approach for querying dynamic OLAP data cubes. In: Kitsuregawa, M., Papazoglou, M.P., Pu, C. (eds.) Proceedings of the 15th International Conference on Data Engineering, Sydney, Austrialia, March 23-26, pp. 328–335. IEEE Computer Society (1999)
45. Gfeller, B., Sanders, P.: Towards optimal range medians. In: [2], pp. 475–486
46. Golynski, A.: Optimal lower bounds for rank and select indexes. Theoret. Comput. Sci. 387(3), 348–359 (2007)
47. Greve, M., Jørgensen, A.G., Larsen, K.D., Truelsen, J.: Cell probe lower bounds and approximations for range mode. In: Abramsky, S., Gavoille, C., Kirchner, C., Meyer auf der Heide, F., Spirakis, P.G. (eds.) ICALP 2010, Part I. LNCS, vol. 6198, pp. 605–616. Springer, Heidelberg (2010)
48. Grossi, R., Gupta, A., Vitter, J.S.: High-order entropy-compressed text indexes. In: SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms), pp. 841–850. ACM/SIAM (2003)
49. Gupta, P., Janardan, R., Smid, M.: Further results on generalized intersection searching problems: Counting, reporting, and dynamization. J. Algorithms 19(2), 282–317 (1995)
50. Hampapuram, H., Fredman, M.L.: Optimal biweighted binary trees and the complexity of maintaining partial sums. SIAM J. Comput. 28(1), 1–9 (1998)
51. Harel, D., Tarjan, R.E.: Fast algorithms for finding nearest common ancestors. SIAM J. Comput. 13(2), 338–355 (1984)
52. He, M., Munro, J.I.: Space efficient data structures for dynamic orthogonal range counting. In: Dehne, F., Iacono, J., Sack, J.-R. (eds.) WADS 2011. LNCS, vol. 6844, pp. 500–511. Springer, Heidelberg (2011)
53. He, M., Munro, J.I., Nicholson, P.K.: Dynamic range selection in linear space. In: [5], pp. 160–169
54. He, M., Munro, J.I., Nicholson, P.K.: Dynamic range selection in linear space. CoRR abs/1106.5076v3 (2013), http://arxiv.org/abs/1106.5076v3
55. He, M., Munro, J.I., Zhou, G.: Path queries in weighted trees. In: [5], pp. 140–149
56. He, M., Munro, J.I., Zhou, G.: Succinct data structures for path queries. In: Epstein, L., Ferragina, P. (eds.) ESA 2012. LNCS, vol. 7501, pp. 575–586. Springer, Heidelberg (2012)
57. Ho, C.T., Agrawal, R., Megiddo, N., Srikant, R.: Range queries in OLAP data cubes. In: Peckman, J.M. (ed.) Proceedings, ACM SIGMOD International Conference on Management of Data: SIGMOD 1997, Tucson, Arizona, USA, May 13-15. SIGMOD Record (ACM Special Interest Group on Management of Data), vol. 26(2), pp. 73–88. ACM Press (1997)
58. Hon, W.K., Shah, R., Thankachan, S.V.: Towards an optimal space-and-query-time index for top-k document retrieval. In: [64], pp. 173–184

59. Hon, W.K., Shah, R., Vitter, J.S.: Space-efficient framework for top-k string retrieval problems. In: 50th Annual IEEE Symposium on Foundations of Computer Science, FOCS 2009, Atlanta, Georgia, USA, October 25-27, pp. 713–722. IEEE Computer Society (2009)
60. Jacobson, G.: Space-efficient static trees and graphs. In: 30th Annual Symp. on Foundations of Computer Science, vol. 30, pp. 549–554 (1989)
61. Jacobson, G.: Succinct Static Data Structures. PhD thesis, Carnegie-Mellon, Technical Report CMU-CS-89-112 (January 1989)
62. Janardan, R., Lopez, M.: Generalized intersection searching problems. Internat. J. Comput. Geom. Appl. 3(1), 39–69 (1993)
63. Jørgensen, A.G., Larsen, K.G.: Range selection and median: Tight cell probe lower bounds and adaptive data structures. In: [83], pp. 805–813
64. Kärkkäinen, J., Stoye, J. (eds.): CPM 2012. LNCS, vol. 7354. Springer, Heidelberg (2012)
65. Karpinski, M., Nekrich, Y.: Searching for frequent colors in rectangles. In: Proceedings of the 20th Annual Canadian Conference on Computational Geometry, Montréal, Canada, August 13-15 (2008)
66. Karpinski, M., Nekrich, Y.: Top-k color queries for document retrieval. In: [83], pp. 401–411
67. Krizanc, D., Morin, P., Smid, M.: Range mode and range median queries on lists and trees. In: Ibaraki, T., Katoh, N., Ono, H. (eds.) ISAAC 2003. LNCS, vol. 2906, pp. 517–526. Springer, Heidelberg (2003)
68. Krizanc, D., Morin, P., Smid, M.H.M.: Range mode and range median queries on lists and trees. Nord. J. Comput. 12(1), 1–17 (2005)
69. Lai, Y., Poon, C., Shi, B.: Approximate colored range and point enclosure queries. J. Discrete Algorithms 6(3), 420–432 (2008)
70. Mäkinen, V., Navarro, G.: Rank and select revisited and extended. Theoret. Comput. Sci. 387(3), 332–347 (2007)
71. Munro, J.I., Spira, P.M.: Sorting and searching in multisets. SIAM J. Comput. 5(1), 1–8 (1976)
72. Munro, J.I.: Tables. In: Chandru, V., Vinay, V. (eds.) FSTTCS 1996. LNCS, vol. 1180, pp. 37–42. Springer, Heidelberg (1996)
73. Muthukrishnan, S.: Efficient algorithms for document retrieval problems. In: [33], pp. 657–666
74. Navarro, G.: Wavelet trees for all. In: [64], pp. 2–26
75. Nekrich, Y.: Orthogonal range searching in linear and almost-linear space. In: Dehne, F., Sack, J.-R., Zeh, N. (eds.) WADS 2007. LNCS, vol. 4619, pp. 15–26. Springer, Heidelberg (2007)
76. Nekrich, Y.: Orthogonal range searching in linear and almost-linear space. Comput. Geom. 42(4), 342–351 (2009)
77. Nekrich, Y., Navarro, G.: Sorted range reporting. In: [36], pp. 271–282
78. Patil, M., Shah, R., Thankachan, S.V.: Succinct representations of weighted trees supporting path queries. J. Discrete Algorithms 17, 103–108 (2012)
79. Patrascu, M.: Succincter. In: Proceedings of the 49th Annual IEEE Symposium on Foundations of Computer Science, Philadelphia, Pennsylvania, USA, October 25-23, pp. 305–313. IEEE (2008)
80. Petersen, H.: Improved bounds for range mode and range median queries. In: Geffert, V., Karhumäki, J., Bertoni, A., Preneel, B., Návrat, P., Bieliková, M. (eds.) SOFSEM 2008. LNCS, vol. 4910, pp. 418–423. Springer, Heidelberg (2008)

81. Petersen, H., Grabowski, S.: Range mode and range median queries in constant time and sub-quadratic space. Inf. Process. Lett. 109(4), 225–228 (2009)
82. Poon, C.K.: Dynamic orthogonal range queries in OLAP. Theoret. Comput. Sci. 296(3), 487–510 (2003)
83. Randall, D. (ed.): Proceedings of the Twenty-Second Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2011, San Francisco, California, USA, January 23-25. SIAM (2011)
84. Sadakane, K.: Succinct representations of *lcp* information and improvements in the compressed suffix arrays. In: [35], pp. 225–232
85. Yao, A.C.C.: On the complexity of maintaining partial sums. SIAM J. Comput. 14(2), 277–288 (1985)